VOLUME XII, NUMBER 4 NOVEMBER/DECEMBER 1990 $6.00

FORTH DIMENSIONS

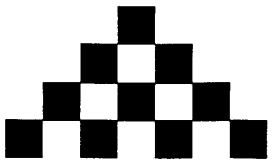
RELIABLE 8086 DIVISION

THREE NUMBER PROBLEM

FORTH ASSEMBLER & LABELS

EASY EXTENDED-PRECISION MATH

68000 NATIVE-CODE FORTH (III)

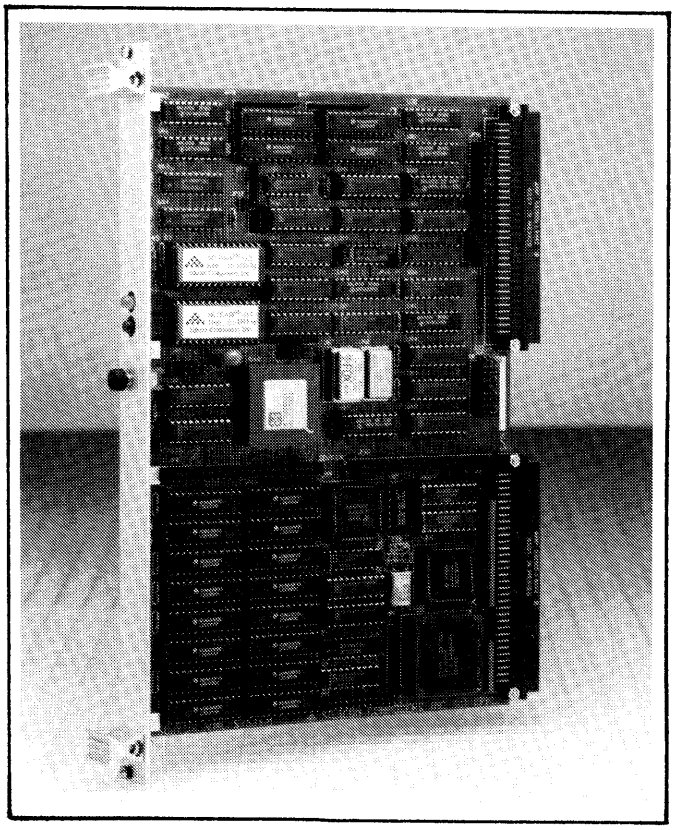


SILICON COMPOSERS INC

Introducing the

SC/FOX™ VME

Single Board Computer



The **SC/FOX VME Single Board Computer** is a full VME slot-one Master/Slave System Controller board. Its speed and features make it an excellent choice for applications such as embedded-systems control, data acquisition, image and numerical processing, process control, factory automation, machine control, inspection systems, robotics, and real-time systems control.

Using the Harris RTX 2000™ real-time Forth chip, the SC/FOX VME hits 18 MIPS at 12 MHz with burst speeds up to 60 MIPS, by executing multiple Forth primitive instructions per clock cycle.

In the U.S. contact:
Silicon Composers
(415) 322-8763
208 California Avenue
Palo Alto, CA 94306

In England contact:
Data Application Ltd
(0285) 651828
16 Dyer Street, Cirencester,
Glos., GL7 2PF England

SC/FOX VME Single Board Computer
- Uses industrial RTX 2000™ microprocessor.
- 16, 20, or 24 MHz input clock operation.
- 32K to 512K bytes 0-wait-state SRAM.
- 64K bytes shadow EPROM.
- 5M byte per second SCSI port.
- Two 56K baud RS232 serial ports.
- Parallel printer port.
- 6U, 233mm by 160mm Eurocard-size board.
- Two 50-pin application headers.
- Optional prototyping plug-on board.

VME Slot-1 System Controller
- Arbitrates bus request 0 to 3.
- Sys-clock and IACK daisy-chain drivers.
- 16-microsecond Bus Timer.

VME Bus Master
- D16/D08(EO) capability.
- Supports bus requests 0 to 3.
- Release-on-request or release-when-done.
- Supports all address modifier codes.
- Word or byte transfers.
- Generates/Handles all 7 VME interrupts, 1-7.
- Release-on-acknowledge interrupter.
- Interrupts on interrupt Ack and bus error.
- AC failure on NMI.

VME Bus Slave
- Maps to full 16M-byte VME address space.
- 128K dual-ported 0-WS static RAM.

SC/Forth Language in EPROM
- Interactive Forth based on Forth-83 Standard.
- Vectored I/O and recursion.
- Supports ASCII text file or block source code.
- 15 level prioritized multitasking.
- Extended control structures.
- Code overlays and 2nd page support.
- Easy turnkey system generation.

RTX 2000 Forth Microprocessor
- Industrial 16-bit CMOS chip in 85-pin PGA.
- 83.33ns machine cycle at 12 MHz system clock.
- 1-cycle instruction execution.
- 1-cycle 16-bit multiply.
- 14-prioritized interrupts, 4-cycle latency.
- 8-channel multiplexed 16-bit I/O bus.
- Two on-chip 256-word stack memories.
- Three on-chip 16-bit timer/counters.

FORTH DIMENSIONS

EDITORIAL

Forth Dimensions
Published by the
Forth Interest Group
Volume XII, Number 4
November/December 1990
Editor
Marlin Ouverson
Advertising Manager
Kent Safford
Design and Production
Berglund Graphics
Circulation/Order Desk
Anna Brereton

MID-LIFE CRISIS

A subject of frequent enquiry among Forth programmers is the future of their language. Is it a prodigy, a pariah, or an idiot savant? Is it dying or just having growing pains? And what are the Forth pioneers doing now—are they churning out C in Unix environments, designing closed systems driven by machine code, or doing the middle-management shuffle? Is it possible to earn a living with Forth any more?

At times Forth seems like someone who works for ten or twenty years, raises a family, faces a few challenges successfully, and then wakes up one morning to ask, "But what do I want out of life? What will I do when I grow up? Is life passing me by?" Past achievements fade, the future is hidden, and a paralyzing uncertainty sets in.

The computing world has changed dramatically over the years and so have the values of its populace. The first microcomputers represented real personal power and freedom wrested from the mainframe priesthood. For a while, everyone with a computer became a programmer to some extent. Eventually, though, the complexity of "micro" systems rivalled or surpassed even that of their bulkier ancestors. Now, most users have no wish to be programmers or to get close to the innards of their machines, and a new kind of priesthood has emerged to minister to their needs.

As the programming horizons expanded, and as the speed and memory constraints of early microcomputers were shattered, Forth's minimalist and relatively austere origins came to seem a little quaint and out of touch to outsiders. Some complained that the computer scientists largely ignored Forth, but many Forth developers ignored computer science, too, except in a wish-you-were-here kind of way. A few engineering departments and educators found Forth to be a useful alternative, and some businesses allowed Forth as a prototyping language. But it seemed that Forth vendors thrived more when they supplemented product sales with consulting and contract programming; even the fiercest of fanatics stopped forecasting a day when Forth would be on every hard drive.

Still, the very scope of modern technology ensures that it is possible to carve out a profitable and enduring niche somewhere, which is the challenge confronting the entire Forth community. To do so will require our every resource, the collective understanding and vision of vendors, academics, professionals, engineers, and computer scientists equally. Like the person stuck in a mid-life crisis, we must start from where we are, not from where we wish we were. The lack of a guaranteed outcome, or the specter of past mistakes, must not prevent us from planning for the future together.

—Marlin Ouverson
Editor

About the Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $24/36 per year by the Forth Interest Group, 1330 S. Bascom Ave., Suite D, San Jose, CA 95128. Second-class postage paid at San Jose, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 8231, San Jose, CA 95155."

Forth-83

RELIABLE 8086 DIVISION

DAVID ARNOLD - KIRKSVILLE, MISSOURI

The Intel 8086 and 80286 microprocessors of the 808x series have some serious, but correctable, bugs in their hardware signed division (the CPU instructions, not the numeric coprocessor functions):

1. Under certain conditions, the 8086 does a division error interrupt when the result would have been valid.
2. The division error interrupt of the 80286 saves the wrong return address.[1]

If an adequate machine code interrupt handler is not in place, division errors could disrupt a program or cause the computer to crash. An important number of computers use one of these processors or a work-alike variant. The IBM PC, XT, and AT; the IBM PS-2 Models 25, 30, and 50; and their many clones are some prominent examples. I'll try to describe the bugs and then show some remedies. The remedies add about 15–50 percent to the execution time of Forth division, and also provide useful information about valid division errors.

The Sample Code

The twenty-four screens of Forth source code include some examples of 808x hardware division and four ways to use it to do Forth-83 division. Screen one, on lines 8–12, determines which demonstration section will be compiled. Just comment out all but the one line that compiles the desired section. The essential source code for each division method occupies about half a dozen screens. It was developed with a Forth-83 system with a Laxen and Perry F83 assembler on a Tandy 1000SX computer (a fairly compatible IBM PC clone with an 8088 CPU). The 80286 bug fixes have been verified and tested slightly on an IBM PS-2 Model 50 and a Tandy TL/2.

808x hardware signed division with the IDIV instruction is demonstrated on screen five. The examples return a flag that shows whether a division error interrupt occurred.

Division method one (screens six through eight) does a Forth division with no error checks. Its speed and size is a baseline for comparison with the more practical methods.

Method two (screens ten through 14) uses a specially installed division error interrupt handler that sets a flag and accommodates the erroneous return address of the 80286. To accommodate the 8086 spurious error, the arithmetic is re-done with general-purpose reliable code if an IDIV instruction caused an interrupt.

Method three (screens 16–19) checks for imminent division error interrupts and takes special action to avoid them.

Applications can work better, and without slow software fixes.

Method four (screens 21–24) uses reliable unsigned division and high-level Forth to synthesize signed division. It's very slow.

When they're compiled, the IDIV demos on screen five and division methods one and two each install a machine code handler for the division error interrupt, INT 0. `BUILD_'INTO` is used once at compile time to fill `'INTO` with the four-byte segment:offset address of the new interrupt handler. After `'INTO` is initialized, `SWAP_INTO` is used to swap the contents of the current system vector with the contents of `'INTO`. Using `SWAP_INTO` a second time restores the saved original vector. The operator should restore the original before leaving Forth.

808x Hardware Division

Hardware integer division with 808x-series processors is very quick. An IBM PC (one of the slowest, with an 8088) can execute a machine code instruction in as little as 35 microseconds. Forth system overhead (stack access, flooring the results, and operating an inner interpreter) and accommodating the division bugs increase that to about 65–100 microseconds. The 80286 processor is even faster.

The IDIV and DIV instructions, respectively, do signed and unsigned division on integers of two basic sizes (Figure One). The signed results are unfloored; a non-zero remainder has the same sign as the dividend. Signed numbers are represented in two's complement form.

If the divisor is zero or if the ideal quotient would be too large to express in the quotient register, the chip does an interrupt through system interrupt vector zero, and the quotient and remainder registers are left undefined.[2] INT 0—which is variously called the division-error, division-exception, division-overflow, or divide-by-zero interrupt—cannot be forestalled by disabling maskable interrupts with the CLI instruction.[3] If an appropriate machine code interrupt handler is not at the location in the vector, the computer will almost surely do something wrong.

An existing interrupt handler might be inadequate or incoherent. MS-DOS provides a simple handler that just displays a "Division Error" message and then termi-

nates the current program. Such a premature exit might bypass essential cleanup procedures, such as finishing and saving work in progress, restoring the display, or restoring various interrupt vectors. An incoherent example is a popup desktop organizer that I use. It leaves the division error vector pointing to machine code that always crashes. (The popup never crashes. Maybe the code is unused or is intended to work in the run-time context of compiled Pascal.)

A program that may cause a division error interrupt (either spurious or true) should install an interrupt handler for itself and should restore the old one before it finishes. If the program is memory resident, it should restore the interrupt before it goes to sleep. A programmer's guide should describe interrupts and ways to change them.

The Division Bugs

Figure Two illustrates the bug in the 8086 processor. If signed division (IDIV) on a 32-bit dividend would produce an ideal quotient of -8000h—or, on a 16-bit dividend, an ideal quotient of -80h—the chip does a spurious division error interrupt.[4] There are some bright spots in this foul-up, though: (1) Only a small portion of possible operands produce the bug, and with a little effort they can be detected beforehand. (2) Unsigned division (DIV) works fine. And (3) if a division error did *not* occur, the quotient and remainder are valid. So there can be no false indications of validity, and reliable methods can either redo cases of possibly spurious interrupts or can avoid them altogether. The task of sifting answers from uncertain results can be fairly simple and straightforward.

Several methods can deal with the 8086 spurious division error:

1. Do nothing about it, if your application would never cause a division error.

2. Install a division error interrupt handler that sets a flag. If the flag is set after a signed division finishes, a reliable general-purpose routine can re-do the problem. The slower code would either confirm the error or return correct results.

3. Check the operands. Avoid true division errors and provide stock results for

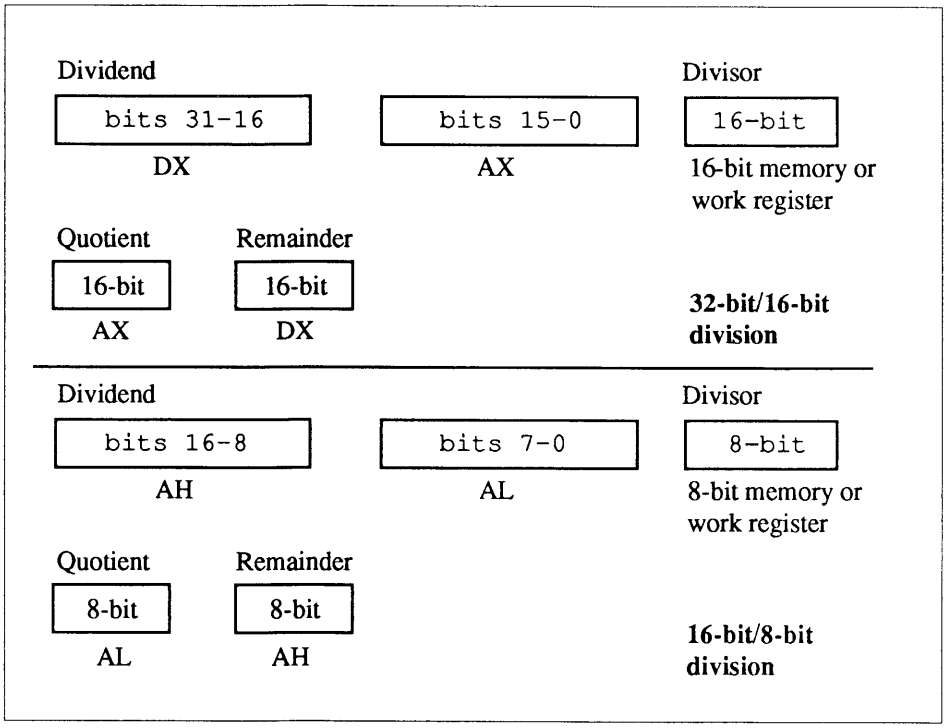
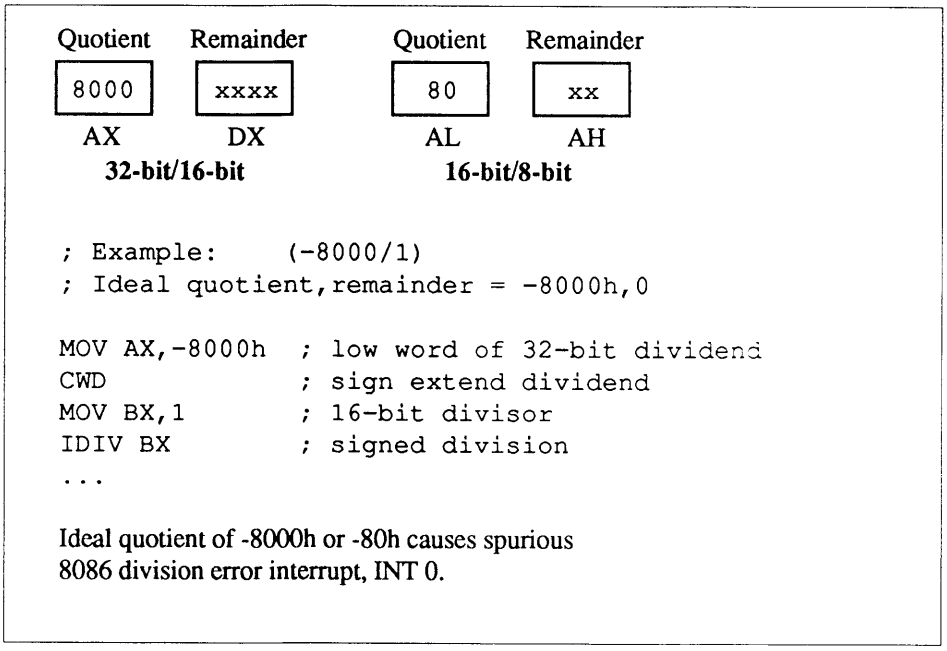


Figure One. 8086 hardware division.




```
; Example:     (-8000/1)
; Ideal quotient,remainder = -8000h,0

MOV AX,-8000h  ; low word of 32-bit dividend
CWD            ; sign extend dividend
MOV BX,1       ; 16-bit divisor
IDIV BX        ; signed division
...
```

Ideal quotient of -8000h or -80h causes spurious 8086 division error interrupt, INT 0.

Figure Two. 8086 DIV bug.

```
ErrorHandle   mov word ptr [div_err?],0FFFFh
              iret
              ...

SampleDiv     mov ax,1234h
              mov dx,5678h
              mov bx,0
              div bx
              ...
```

INT 0 should save this address.
80286 saves this address instead.

after INT 0	after IRET	—stack—
...	...	
xxx	SP → xxx	
CPU FLAGS	cpu flags	
CS	cs	
SP → IP	ip	
...	...	

Division error interrupt saves CPU flags and return address on the stack and jumps to ErrorHandle.

IRET restores flags from the stack and jumps to return address (CS:IP).

Figure Three. 80286 return address bug.

UM/MOD	32-bit unsigned dividend 16-bit unsigned divisor
/MOD / MOD	16-bit signed dividend 16-bit signed divisor
*/MOD */ intermediate results	32-bit signed dividend 16-bit signed divisor
2/	16-bit signed division by two

All results are floored.
(Remainder has the same sign as the divisor.)

Figure Four. Forth-83 division.

cases of imminent spurious error.

4. Synthesize signed division with the properly functioning unsigned division.

5. Do long division with traditional bit-by-bit looping machine code.

6. Use an 8087 math coprocessor or software emulation of an 8087.

The first method might work in a program that's used only in particular hardware for a special application. The second and third methods are reliable and fairly quick. The fourth and fifth are generally slow, but are adaptable to almost any extent. The sixth is expensive in hardware, and is slow in software emulation.

Figure Three illustrates the incorrect return address saved by the 80286 division error interrupt. It should push onto the stack the address of the instruction that follows the errant divide instruction. Instead, the address of the division instruction itself is pushed onto the stack.[5] An IRET under that condition would just return to the instruction that caused the interrupt. The dividend and divisor are left unchanged on all the machines I've tested,[6] and a never-ending loop ensues.

Here are some methods for dealing with the 80286 bug:[7]

1. Do nothing about it if division error never occurs.

2. Install a special interrupt handler that reaches into the stack and bumps the saved return address past the division instruction.

3. Install a special interrupt handler that loads the dividend and divisor with dummy values that do not cause an error. Upon return from the interrupt, a superfluous division is performed on the dummies and then execution continues.

The first method might work in hardware that never causes errors.

If the second method could determine whether the current processor were an 80286 (and not an 8086 or 80186), it could use a table of opcodes and corresponding instruction lengths to adjust the return ad-

dress. If a few NOPs were compiled after the division instruction, the return address could be bumped a constant amount on any processor. The NOPs would waste time, though.[8]

The third method would work if the divisor were in a known location and would be given a non-zero value. When a division error occurs the dividend (in the CPU) is left undefined, so nothing would be lost by changing it. If the divisor were in ROM or in a permanent RAM-based table and couldn't be changed, it presumably would never be zero. In that case, just zeroing the dividend would be sufficient.

Figure Four lists the four types of integer division required in a Forth-83 system. Forth-83 also requires that division results be floored. I'll say something about that, and then describe some reliable division methods.

Floored and Unfloored Division

School children are taught floored division. The remainder has the sign of the dividend. If the dividend and divisor have different signs (and if the quotient is non-zero), the quotient is negative; otherwise, it is positive. This is a natural way to work with sizes of things. It's a fairly easy way to do hand-worked long division, and the unfloored remainder is a natural starting point for further division to increase the precision of the quotient with additional digits. Unfloored quotients, positive or negative, have an average bias of 0.5 toward zero.

The floored remainder has the sign of the divisor. The sign of the quotient is like that of unfloored division. However, it turns out that whenever the signs of the dividend and divisor differ, the floored quotient is always non-zero. Floored quotients have an average bias of 0.5 in the negative direction.[9]

Unfloored results are easily produced by adjusting the signs of the quotient and remainder of unsigned division. Floored results can be produced by simple adjustments of the unfloored quotient and remainder (Figure Five).

Floored and unfloored results are not two different answers for the same division. They both describe the same thing; they just express it differently. For example, look at the problem in Figure Six. The dividend and divisor are 3 and -5. The unfloored quotient and remainder are 0 and

		unfloored		floored		
dividend	**divisor**	**q**	**r**	**q'**	**r'**	
+	+	+	+	+	+	
		uq	ur	q	r	
+	-	-	+	-	-	
		-uq	ur	q	r	[if r=0]
				q-1	r+dvs	[if r<>0]
-	+	-	-	-	+	
		-uq	-ur	q	r	[if r=0]
				q-1	r+dvs	[if r<>0]
-	-	+	-	+	-	
		uq	-ur	q	r	

uq, ur = unsigned quotient,remainder of abs(dividend)/abs(divisor)

Figure Five. Floored/unfloored division conversions.

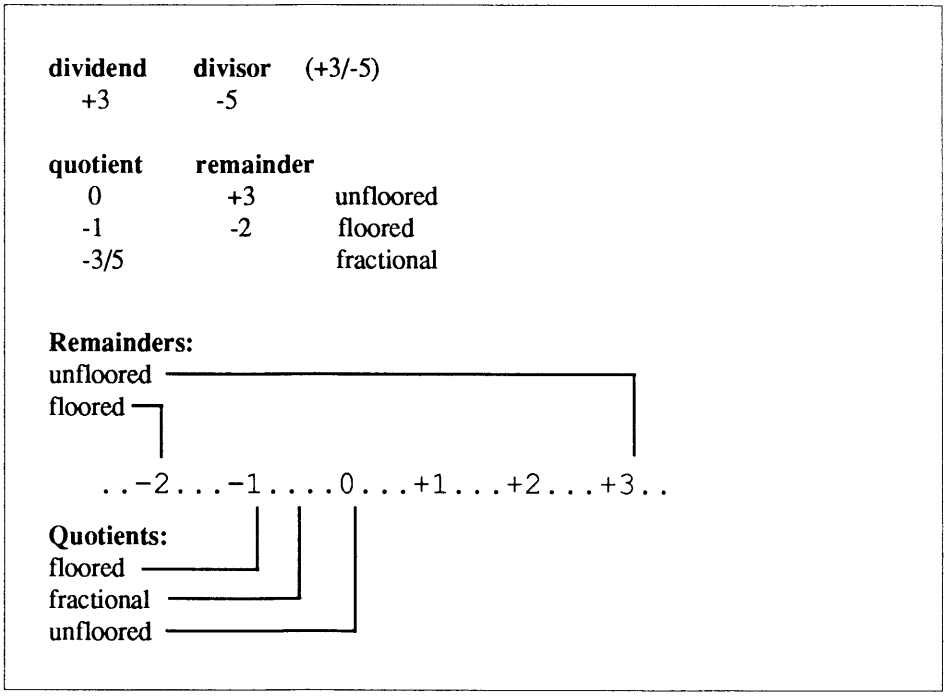


Figure Six. Floored/unfloored example.

Method	UM/MOD	/MOD	*/MOD	Size
1	61 µs	67 µs	101 µs	324 bytes
2	69	98	139	473
3	73	101	145	277
4	73	1.71 ms	2.07 ms	448
	2/	**(M/MOD)**	**M/MOD**	**M***
2 & 3	27 µs			
4		1.01 ms	1.45 ms	0.43 ms

Method 1: No error handling
Method 2: Recalculates after low-level errors
Method 3: Avoid division error interrupts
Method 4: High-level signed division

Indirect-threaded code Forth system.
IBM PC clone with 8088 CPU at 4.77 MHz.

Sample timing loops:

```
0 0 do    i 7 11    3drop  loop          (idle)
0 0 do    i 7 11    */mod  2drop  loop   (test)
```

(time/division) = (test-idle)/65536

Figure Seven. Division statistics.

3. The floored quotient and remainder are -1 and -2. The fractional quotient is -3/5, with the fractional remainder implicitly zero. In each case, multiply the quotient by the divisor and add the remainder: the result is the dividend. Notice that rounding the fractional quotient either left or right to the nearest whole number results in either the floored or unfloored quotient, respectively.

Four Division Methods

Here are four ways to do Forth division with 808x division. Method One doesn't accommodate division errors, and is included for comparison with more reliable methods. Methods Two and Three are practical 8086 code implementations of Forth-83 division that could be transplanted directly into a Forth system. Method Four is mostly in high-level Forth and might work well on speedy Forth engines.

Division Error Flag

The methods that accommodate division error (Methods Two, Three, and Four) leave their error status in `DIV_ERR?`—false if the results are valid, true if they are not. Error or not, the same number of items is left on the stack. No other error action is taken.

Low-level division begins by clearing `DIV_ERR?` with a `FALSE` flag. High-level division clears `DIV_ERR?` by using the low-level word `UM/MOD`. If the divisor is zero or if quotient overflow occurs during division, a `TRUE` flag is left in `DIV_ERR?`. If quotient overflow occurs during unfloored-to-floored conversion, `DIV_ERR?` is updated with a `TRUE` flag. If a spurious division error interrupt occurs, `DIV_ERR?` momentarily contains `TRUE`, but is immediately cleared and updated when the auxiliary code re-does the problem.

Flooring Conversion

All floored results are obtained by doing unfloored division and then converting the results. If the signs of the dividend and divisor differ and if the remainder is non-zero, the divisor is added to the unfloored remainder and the unfloored quotient is decremented by one. If the quotient changed sign when it decremented, it overflowed the word size. The single-precision `CODE` operator `/MOD` doesn't check for flooring conversion overflow because it can't occur there. (Its greatest negative quotient is the result of -8000h/1, which has a remainder of zero and, therefore, does not require adjustment of the quotient.)

Method One

No error checks
Screens Six through Eight

Method One demonstrates nearly the greatest possible speed of Forth division with the 808x series (I think). It does no error checking, provides no error status flag, and isn't reliable. A dummy interrupt handler is installed to keep the computer from crashing when a division error interrupt occurs. Floored results are produced. The method is basically Method Two without error handling.

Method Two

Recalculate possibly spurious error
Screens Ten through 14

Method Two accommodates the 8086 spurious division error and the 80286 erroneous return address, and takes advantage of nearly the full speed of 808x division. First, the flag in `DIV_ERR?` is cleared, then division is performed with `DIV` or `IDIV`. If a division error interrupt occurs, a specially installed interrupt handler posts a `TRUE` flag in `DIV_ERR?` and loads the dividend and divisor with dummy values that cannot cause a division error interrupt—the dividend (DX:AX) is loaded with zero and the divisor (BX) with one. (Any non-zero dummy divisor would do.)

On 80286 machines, a superfluous division takes place immediately upon return from an interrupt, and then execution continues. If `DIV_ERR?` is found to have been set, an 8086 may have caused a spurious interrupt. The arithmetic is re-done with reliable general-purpose code that either confirms the error or returns correct results. The final error status is left in `DIV_ERR?`.

The slight extra effort of installing an interrupt handler might be more than offset by the extra speed and simplicity of arithmetic routines that don't need to stand on their heads to avoid division error inter-

rupts. Unless a program were peculiarly attached to interrupt-causing operands, not much time would be used re-doing the arithmetic.

UM/MOD
DIV does the dividing. If the division error flag is set, a true division error occurred.

/MOD / MOD */MOD */
The IDIV instruction does the division. Then, if the division error flag was set by an interrupt, SAFE_M/MOD is used to either confirm the error or to return correct answers. The standard Forth division words would do a machine code jump directly to the code body of SAFE_M/MOD, which would continue just as if it had been called directly. SAFE_M/MOD is about the same as */MOD of Method Three, and avoids division error interrupts altogether.

2/
This signed division by two is done with an 808x arithmetic shift-right instruction, SAR. All bits are shifted right, and the sign bit is filled with the same value that the old sign bit held. It directly produces a floored result, and it can't overflow the quotient word size.

Method Three
Detect imminent division error
Screens 16–19

Method Three checks for an imminent division error interrupt and provides its own error action. If a spurious 8086 error is about to occur, a stock answer is provided. If a true error is imminent, the problem is abandoned and a division error is flagged. The division error status is left in DIV_ERR?.

UM/MOD
This is nicely handled by the reliable DIV instruction. If the high word of the dividend is equal to or greater than the divisor, division by zero or quotient overflow is imminent.

/MOD / MOD
These each have one case that would spuriously cause an 8086 division error interrupt, -8000h divided by 1. That special case can be swiftly and easily found before doing any division, and a stock result provided. Two cases would cause a true error, either a zero divisor or -8000h/-1. The tests for imminent division error interrupt are a little intricate. Notice in screen 16, line seven, that if the case -8000h/1 is found, execution falls through with the proper quotient and remainder in registers AX and DX.

*/MOD */
These are tougher. Myriad intermediate double-precision results could cause quotient overflow, either spurious or true, and I don't know of a simple inspection of signed numbers that could spot them beforehand. The method I've used converts the operands to their absolute values and then uses unsigned division (DIV).

2/
This is done with an 808x shift arithmetic right, SAR, and is foolproof. (See 2/ of Method Two.)

Method Four
High-level division
Screens 21–24

Method Four uses reliable unsigned division and high-level Forth to synthesize signed division. It leaves its division error status in DIV_ERR?.

UM/MOD does low-level unsigned division with a check for an imminent division error. (M/MOD), FLOOR_QR, and M/MOD together perform high-level signed division. (M/MOD) does unfloored division, with a check for quotient overflow. FLOOR_QR converts an unfloored quotient and remainder to floored form, with a check for quotient overflow. M/MOD uses the two preceding words to do floored division. The Forth standard division words use this basic toolkit.

Division error can occur at several nesting levels as a dividend and divisor grind through these words. /MOD, for example, calls M/MOD, which calls FLOOR_QR and (M/MOD), which calls UM/MOD; and most of the subordinate words can have a division error. Rather than immediately abort computation at some indeterminate level, they take no action other than posting an error flag in DIV_ERR?. If a division error takes place early, later steps will be working with meaningless numbers. They won't crash, though, and DIV_ERR? can provide the last word on validity of the results.

Conclusions

Method One (no error checks) is somewhat faster than the others and might work well in a hardware project that would never provide troublesome operands. Method Two (re-do possible error) is quick. It needs a special interrupt handler, a divisor that is known to be non-zero or that can be safely altered, and redundant auxiliary code. Method Three (avoid error) is a bit slower and is more portable. It doesn't juggle the host computer interrupt vectors, nor does it unpredictably alter the divisor. Method Four is completely portable to any Forth system that has unsigned division.

Figure Seven lists some size and speed statistics for the various division methods. Some specially coded equal-duration 3DROP and 2DROP definitions provided consistent stack clean-up for the empty-loop and test-loop times of UM/MOD and */MOD (which receive three 16-bit numbers but leave two).

These remarkable speeds were obtained on a pretty ordinary PC clone with an 8088 microprocessor. 80286-based computers, with which PCs share much software, are still faster. Applications that use a lot of integer division on these machines might be made to work better than we have expected if they accommodate the 808x division bugs and don't resort to slow software fixes.

Bibliography

Intel. *iAPX 86, 88, 186, and 188 User's Manual Programmer's Reference*, Intel Corporation, 1985.

The Intel programmer's reference books are nearly essential for doing much 808x programming. Some of the machine code instructions affect the processor flags in quirky ways, and some have optimized forms for using the accumulator registers (AX and AL). The section that describes the instructions seems fairly clear and complete, and provides many useful examples. Available from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, California 95051

International Business Machines Corporation. *Technical Reference Personal Computer AT*, Part number 6280070.

For a catalog of IBM technical references, write to:
IBM Technical Directory
P.O. Box 2009 *(Continued on page 40.)*

```
Screen# 1          Forth-83
0 ( 8086 division                                    02May90dna )
1 only   forth definitions  also   decimal
2 create XMARK                              ( FORGET'able marker )
3 vocabulary XDIV   xdiv also
4   cr .( utils. )                          2  dup u. load
5   here                    ( Start address of division routines )
6 create DIV_ERR?  0 ,
7
8 \ cr .( 808x division examples. )         5 5    thru
9 \ cr .( Div method 1.  No checks. )       6 8    thru
a \ cr .( Div method 2.  Redo error. )      10 14  thru
b \ cr .( Div method 3.  Avoid error. )     16 19  thru
c   cr .( Div method 4.  High level. )      21 24  thru
d   cr .( Div size : )  here swap -  u.
e
f only   forth definitions  also  xdiv also   decimal
-----
Screen# 2          Forth-83
0 ( system utils                                     28Mar90dna )
1 : \    ( -- )  >in @   63 +  dup 64 mod  -   >in ! ;  immediate
2 : \S   ( -- )  1024 >in ! ;
3 : THRU ( first last -- )
4  over over 1+  u< if   1+ swap do  i u.  i load  loop   then ;
5
6 : S>D   ( n -- d )  dup 0< ;              ( uses Forth-83 TRUE )
7 : DABS  ( d -- d' )  dup 0< if  dnegate  then ;
8 : 2DUP  ( n1 n2 -- n1 n2 n1 n2 )  over over ;
9 : 2DROP ( n n -- )  drop drop ;
a : NIP   ( n1 n2 -- n2 )  swap drop ;
b
c 0 0=  constant TRUE
d 1 0=  constant FALSE
e
f ( some definitions assume that Forth TRUE = FFFFh = -1 )
-----
Screen# 3          Forth-83
0 hex ( 808x division error interrupt handler          02May90dna )
1 create 'INT0  4 allot                            ( 808x  seg:ofs )
2
3 code NEW_INT0
4  0FFFF # div_err? #) cs: mov       ( div_err? = true )
5  ax ax xor  cwd   1 # bx mov       ( safe 80286 dummy operands )
6  iret
7 end-code   decimal
8
9 4 load       ( install/restore division error interrupt vector )
a
b build_'int0  ( initialize seg:ofs in 'INT0 )
c swap_int0    cr .( 8086 division error interrupt installed. )
d
e
f
```

```
Screen# 4          Forth-83
0 hex ( install/restore div err interrupt handler      02May90dna )
1 code SWAP_INT0  ( -- )
2  ax push  bx push  cx push  dx push  si push  di push  bp push
3  es push   ds push
4  3500 # ax mov  21 int                  ( .. -- ES:BX=seg:ofs )
5  ds pop   es push  bx push   ds push    ( .. -- .. seg ofs seg )
6  'int0 #) dx lds                        ( .. -- DS:DX=seg:ofs )
7  2500 # ax mov  21 int
8  ds pop   'int0 #) pop  'int0 2+ #) pop
9  es pop  bp pop  di pop  si pop  dx pop  cx pop  bx pop  ax pop
a  next
b end-code
c
d code BUILD_'INT0  ( -- )
e  ' new_int0 >body # 'int0 #) mov   cs 'int0 2+ #) mov   next
f end-code   decimal
-----
Screen# 5          Forth-83
0 ( 808x hardware division examples                    02May90dna )
1 3 load      ( Compile & install 8086 division error interrupt. )
2
3 code 16IDIV  ( dvd_lo dvd_hi dvs -- rem quo t=err )
4  ax ax xor  ax div_err? #) mov                    ( clear flag )
5  bx pop  dx pop  ax pop   bx idiv
6  dx push  ax push   div_err? #) push
7  next
8 end-code
9
a code 8IDIV ( b_dvd_lo b_dvd_hi b_dvs -- b_rem b_quo t=err )
b  ax ax xor  ax div_err? #) mov
c  bx pop   dx pop  ax pop  dl ah mov   bl idiv
d  bx bx xor  ah bl mov  bx push   al bl mov  bx push
e  div_err? #) push   next
f end-code
-----
Screen# 6          Forth-83
0 ( 8086 division Method 1  No error checks  /mod      02May90dna )
1 3 load      ( Compile & install 8086 division error interrupt. )
2
3 code /MOD  ( dvd dvs -- rem quo )
4  bx pop  ax pop
5  cwd                          ( DX:AX=d_dvd BX=dvs )
6  dx cx mov   bx idiv          ( CX=dvd_sgn  AX,DX=quo,rem )
7  bx cx xor  0< if             ( dvd & dvs sgns differ? )
8    dx dx or  0<> if           ( rem nonz? )
9      ax dec  bx dx add   then  then   ( floor quo,rem )
a  dx push  ax push   next
b end-code
c \s
d No checks for division error or spurious interrupts.
e Dummy division error interrupt handler installed to keep
f computer from crashing through an undefined interrupt vector.
```

```
Screen# 7         Forth-83
0 ( 8086 divi  No error checks  */mod                02May90dna )
1 code */MOD  ( n1 n2 n3 -- rem quo )
2  bx pop  ax pop  dx pop
3  dx imul                        ( DX:AX=d_dvd BX=dvs )
4  dx cx mov   bx idiv            ( CX=dvd_sgn  AX,DX=quo,rem )
5  bx cx xor  0< if               ( dvd & dvs sgns differ? )
6    dx dx or  0<> if             ( rem nonz? )
7      ax dec  bx dx add   then  then   ( floor quo,rem )
8  dx push  ax push   next
9 end-code
a
b
c
d
e
f
-----
Screen# 8         Forth-83
0 ( 8086 divi  No error checks  um/mod  2/           02May90dna )
1 code UM/MOD  ( ud_dvd u_dvs -- urem uquo )
2  bx pop  dx pop  ax pop
3  bx div
4  dx push  ax push   next
5 end-code
6
7 code 2/  ( n -- n/2 )
8  ax pop   ax sar   ax push  next
9 end-code
a
b : /   ( dvd dvs -- quo )  /mod nip ;
c : MOD ( dvd dvs -- rem )  /mod drop ;
d : */  ( n1 n2 n3 -- quo )  */mod nip ;
e
f
-----
Screen# 9         Forth-83
0 ( Forth division Method 2 notes  Install interrupt  05Apr90dna )
1 8086 division error interrupt handler just sets a flag.
2
3 Low level Forth division clears division error flag and uses
4 8086 DIV, IDIV, & SAR instructions.
5
6 UM/MOD and 2/ leave results on stack & take no error action.
7
8 Signed division uses IDIV & checks error flag.  If flag is set,
9 arithmetic redone with slow & reliable word, which either
a confirms error or returns correct results.
b
c IF DIV_ERR? is now TRUE, true division error occurred, and
d division results on stack are invalid.  Final stack depth is
e same, whether error or not.
f
```

```
Screen# 10        Forth-83
0 ( 8086 div2  INT 0  safe_m/mod 1 of 2              02May90dna )
1 3 load      ( Compile & install 8086 division error interrupt. )
2 hex
3 code SAFE_M/MOD  ( d_dvd dvs -- rem quo )
4  bx pop  dx pop  ax pop
5  0FFFF # div_err? #) mov        ( default flag = true )
6  dx di mov  bx cx mov           ( keep dvd_sign & dvs )
7                                 ( dvd & dvs to absolute values )
8  dx dx or  0< if   dx dx xor  ax neg  di dx sbb   then
9  bx bx or  0< if   bx neg   then
a ( cont. )
b decimal             ( DX:AX=ud_dvd BX=u_dvs  DI=dvd_sgn CX=dvs )
c
d
e
f
-----
Screen# 11        Forth-83
0 hex ( 8086 div2  safe_m/mod 2 of 2                 02May90dna )
1  bx dx cmp  u< if         ( imminent overflow? )
2    bx div                 ( AX,DX=uquo,urem  DI=dvd_sgn CX=dvs )
3    di di or  0< if  dx neg  then       ( dvd neg?  rem = -urem )
4    cx di xor  0< if       ( dvd & dvs signs differ? )
5      ax neg               ( quo = -uquo. )
6      dx dx or  0<> if  ax dec  cx dx add  then ( floor quo,rem )
7    then                   ( AX,DX=quo',rem'  DI=quo_sgn )
8    ax di xor  0>= if      ( quotient not overflow? )
9      0 # div_err? #) mov  ( ok, flag = false )
a    then
b  then
c  dx push  ax push
d  next
e end-code
f decimal
-----
Screen# 12        Forth-83
0 ( 8086 div2  /mod                                  02May90dna )
1 code /MOD  ( dvd dvs -- rem quo )
2  ax ax xor  div_err? #) ax mov   ( clear flag )
3  bx pop  ax pop
4  cwd                            ( DX:AX=d_dvd BX=dvs )
5  ax push  dx push  bx push      ( keep division operands )
6  dx cx mov   bx idiv            ( CX=dvd_sgn  AX,DX=quo,rem )
7  bx cx xor  0< if               ( dvd & dvs sgns differ? )
8    dx dx or  0<> if             ( rem nonz? )
9      ax dec  bx dx add   then  then   ( floor quo,rem )
a  0 # div_err? #) cmp            ( Error? Redo with safe word. )
b  0<> if   ' safe_m/mod >body #) jmp   then
c  6 # sp add   dx push  ax push   next
d end-code
e
f
```

```
Screen# 13        Forth-83
0 hex ( 8086 div2  */mod                              02May90dna )
1 code */MOD  ( n1 n2 n3 -- rem quo )
2  ax ax xor  ax div_err? #) mov   ( clear flag )
3  bx pop  ax pop  dx pop
4  dx imul                        ( DX:AX=d_dvd BX=dvs )
5  ax push  dx push  bx push      ( keep division operands )
6  dx cx mov   bx idiv            ( CX=dvd_sgn  AX,DX=quo,rem )
7  bx cx xor  0< if               ( dvd & dvs sgns differ? )
8    dx dx or  0<> if             ( rem nonz? )
9      ax dec  bx dx add   then  then   ( floor quo,rem )
a  ax cx xor                      ( quot overflow? )
b  0< if  0FFFF # div_err? #) mov  then
c  0 # div_err? #) cmp            ( Error? Redo with safe word. )
d  0<> if   ' safe_u/mod >body #) jmp   then
e  6 # sp add   dx push  ax push   next
f end-code   decimal
-----
Screen# 14        Forth-83
0 ( 8086 div2  /  mod  */  um/mod  2/              02May90dna )
1 : /   ( dvd dvs -- quo )  /mod nip ;
2 : MOD ( dvd dvs -- rem )  /mod drop ;
3 : */  ( n1 n2 n3 -- quo )  */mod nip ;
4
5 code UM/MOD  ( ud_dvd u_dvs -- urem uquo )
6  ax ax xor  ax div_err? #) mov                ( clear flag )
7  bx pop  dx pop  ax pop
8  bx div
9  dx push  ax push   next
a end-code
b
c code 2/  ( n -- n/2 )
d  ax ax xor  ax div_err? #) mov   ax pop  ax sar  ax push   next
e end-code
f
-----
Screen# 15        Forth-83
0 ( Forth division Method 3 notes  Avoid interrupt    05Apr90dna )
1 CODE words check operands to avoid 8086 division error
2 interrupt, and use 8086 DIV and IDIV.  Division results left on
3 stack, and division error status left in DIV_ERR? .
4
5 Division error avoidance checks --
6   Check for imminent spurious 8086 division error.
7     If so, provide stock answer.
8   Check for imminent true overflow.
9     If so, abandon computation.
a     If not, continue and leave computed answer.
b
c Final stack depth is same, whether error or not.
d
e
f
```

```
Screen# 16        Forth-83
0 hex ( 8086 div3  /mod                               02May90dna )
1 code /MOD  ( dvd dvs -- rem quo )
2  bx pop  ax pop
3  di di xor  di dec                      ( default flag = true )
4  bx bx or  0<> if                         ( dvs=0? )
5    ax cx mov  bx dx mov
6    8000 # cx xor  dx inc  cx dx or  0<> if ( dvd=-8000 dvs=-1? )
7      bx dx mov  dx dec  cx dx or  0<> if   ( dvd=-8000 dvs=1? )
8        cwd  dx cx mov   bx idiv            ( signed division )
9        bx cx xor  0< if   dx dx or  0<> if ( quo neg? rem nonz?)
a            ax dec  bx dx add   then  then  ( floor quo,rem )
b      then
c      di inc                               ( ok, flag = false )
d    then
e  then   dx push  ax push   di div_err? #) mov   next
f end-code   decimal
-----
Screen# 17        Forth-83
0 hex ( 8086 div3  */mod  1 of 2                      28Apr90dna )
1 code */MOD  ( n1 n2 n3 -- rem quo )
2  bx pop  dx pop  ax pop
3  0FFFF # div_err? #) mov        ( default flag = true )
4  dx imul                        ( signed multiply )
5  dx di mov  bx cx mov           ( keep dvd_sign & dvs )
6                                 ( dvd & dvs to absolute values )
7  dx dx or  0< if   dx dx xor  ax neg  di dx sbb   then
8  bx bx or  0< if   bx neg   then
9 ( cont. )
a decimal             ( DX:AX=ud_dvd BX=u_dvs  DI=dvd_sgn CX=dvs )
b
c
d
e
f
-----
Screen# 18        Forth-83
0 hex ( 8086 div3  */mod  2 of 2                      28Apr90dna )
1  bx dx cmp  u< if         ( imminent overflow? )
2    bx div                 ( AX,DX=uquo,urem  DI=dvd_sgn CX=dvs )
3    di di or  0< if  dx neg  then       ( dvd neg?  rem = -urem )
4    cx di xor  0< if       ( dvd & dvs signs differ? )
5      ax neg               ( quo = -uquo. )
6      dx dx or  0<> if  ax dec  cx dx add  then ( floor quo,rem )
7    then                   ( AX,DX=quo',rem'  DI=quo_sgn )
8    ax di xor  0>= if      ( quotient not overflow? )
9      0 # div_err? #) mov  ( ok, flag = false )
a    then
b  then
c  dx push  ax push
d  next
e end-code
f decimal
```

```
Screen# 19        Forth-83
0 ( Forth div3  / mod  */  2/                        28Apr90dna )
1 : /      ( dvd dvs -- quo )      /mod  nip ;
2 : MOD    ( dvd dvs -- rem )      /mod  drop ;
3 : */     ( n1 n2 n3 -- quo )     */mod  nip ;
4
5 code UM/MOD  ( ud_dvd u_dvs -- urem uquo )
6  bx pop  dx pop  ax pop
7  di di xor  di dec                    ( default flag = true )
8  bx dx cmp  u< if                     ( no imminent overflow? )
9    bx div   di inc   then             ( ok, flag = false )
a  dx push  ax push   di div_err? #) mov   next
b end-code
c
d code 2/  ( n1 -- n2 )
e  ax ax xor  ax div_err? #) mov   ax pop  ax sar  ax push   next
f end-code
-----
Screen# 20        Forth-83
0 ( Forth division Method 4 notes  High level        05Apr90dna )
1 UM/MOD does all low level division.
2
3 UM/MOD clears DIV_ERR? and checks operands for imminent division
4 error.  If division error imminent, arithmetic is abandoned and
5 TRUE is posted in DIV_ERR?.
6
7 All signed division done at high level.  If quotient overflows
8 during flooring conversion, TRUE is posted in DIV_ERR?.
9
a IF DIV_ERR? is TRUE when division done, division results on
b stack are invalid.  Final stack depth is same, whether error or
c not.
d
e
f
-----
Screen# 21        Forth-83
0 ( 8086 div4  um/mod  utils                         28Apr90dna )
1 code UM/MOD  ( ud_dvd u_dvs -- urem uquo )
2  bx pop  dx pop  ax pop
3  di di xor  di dec                    ( default flag = true )
4  bx dx cmp  u< if                     ( no imminent overflow? )
5    bx div
6    di inc                             ( ok, flag = false )
7  then
8  dx push  ax push   di div_err? #) mov
9  next
a end-code
b
c : ?NEGATE     ( n sgn -- n' )  0< if  negate  then ;
d : ?DNEGATE    ( d sgn -- d' )  0< if  dnegate  then ;
e : -ALIKE?     ( sgn1 sgn1 -- t=signs_differ )  xor 0< ;
f : SET_DIV_ERR ( -- )  true div_err? ! ;
```

```
Screen# 22        Forth-83
0 ( Forth div4  unfloored division                   02May90dna )
1 : (M/MOD)  ( d_dvd n_dvs -- rem quo )
2  over >r  2dup xor >r           ( -r- dvd_sgn quo_sgn )
3  >r  dabs  r> abs               ( .. -- ud_dvd u_dvs )
4  um/mod                         ( .. -- urem uquo )
5  r@ ?negate                     ( .. -- .. quo )
6  dup r> -alike?  over and       ( uquo maybe too big? & nonz? )
7  if  set_div_err  then          ( flag division error )
8  swap  r> ?negate  swap ;       ( .. -- rem quo )
9
a
b
c \s  If final value of quotient is non-zero and its sign differs
d from nominal computed sign, quotient overflowed wordsize.
e Overflow can occur during unsigned division, too.
f Flag in DIV_ERR? is last word on validity of division results.
-----
Screen# 23        Forth-83
0 ( Forth div4  floor division results               05Apr90dna )
1 : FLOOR_QR  ( rem quo dvs dvd_sgn -- rem' quo' )
2  over xor                           ( .. -- .. quo_sgn )
3  dup 0<  4 pick  and  if            ( quo_sgn neg? rem nonz? )
4    >r                               ( -r- quo_sgn )
5    rot +  swap  1-                  ( .. -- rem' quo' )
6    dup r> -alike?                   ( quot overflow? )
7    if  set_div_err  then            ( flag division error )
8    2dup
9  then                               ( .. -- rem' quo' x x )
a  2drop ;
b \s  Receive unfloored remainder & quotient, divisor, & sign of
c dividend.  Compute nominal sign of quotient.  If nominal
d quotient negative & remainder non-zero, adjustment needed.  Add
e divisor to remainder, & decrement quotient.
f If quotient sign now unlike nominal sign, quotient underflowed.
-----
Screen# 24        Forth-83
0 ( Forth div4  mixed precision  Forth division      05Apr90dna )
1 : M*     ( n1 n2 -- d )
2  2dup xor >r  swap abs swap abs     ( .. -- u1 u2  -r- pro_sgn )
3  um*
4  r> ?dnegate ;
5
6 : M/MOD ( d_dvd dvs -- rem quo )
7  over >r  dup >r           ( .. -- d_dvd dvs  -r- dvd_sgn dvs )
8  (m/mod)   r> r> floor_qr ;
9
a : /MOD  ( dvd dvs -- rem quo )  >r  s>d  r>  m/mod ;
b : /     ( dvd dvs -- quo )  /mod nip ;
c : MOD   ( dvd dvs -- rem )  /mod drop ;
d : */MOD ( n1 n2 n3 -- rem quo )  >r  m*  r>  m/mod ;
e : */    ( n1 n2 n3 -- quo )  */mod nip ;
f : 2/    ( n -- n/2 )  2 / ;
```

EXTENDED-PRECISION MATH MADE EASY

DOUGLAS ROSS - GREENBELT, MARYLAND

In order to do extended-precision math easily, it is necessary to keep track of the carry that is generated when two cells are added or subtracted. This carry must then be applied to the next level of precision for the subsequent partial word length operations.

Though it is possible to do extended-precision math by synthesizing carries using double-word operators, that approach is cumbersome. It also produces code that is not intuitively obvious to understand and which runs much more slowly than necessary.

Your extended-precision math will be greatly enhanced.

The capability of doing extended-precision math can be greatly enhanced with the inclusion of six simple words into your Forth. These words can added to any Forth system with minimal effort. However, their use will greatly enhance the capability to produce extended-precision math operators that are easy to understand, and that are much faster in their operation than those created by other approaches.

Getting Carried Away

One of the key features of Forth is the ability it allows the programmer to get at the "machine" more directly than any language except assembly language. However, one glaring omission in Forth is its inability to allow the programmer to directly get at the carry in high-level Forth.

```
: D+  ( al ah bl bh -- l h )
  >R SWAP >R   CLRC
  +c  R> R>  +c  ;

: D-  ( al ah bl bh -- l h )
  >R SWAP >R   SETC
  -c  R> R>  -c  ;
```

Figure One. New definitions for `D+` and `D-`.

```
: T+  ( al am ah bl bm bh -- l m h )
  CLRC  >R >R SWAP  >R SWAP >R
  +c R> R> R> SWAP  >R
  +c R> R> +c  ;

: T-  ( al am ah bl bm bh -- l m h )
  SETC  >R >R SWAP  >R SWAP >R
  -c R> R> R> SWAP  >R
  -c R> R> -c  ;
```

Figure Two. These words perform a(l,m,h) +/- b(l,m,h) with carry/borrow.

```
: T+  ( al am ah bl bm bh -- l m h )
  >R >R SWAP  >R SWAP >R
  0 SWAP 0  D+
  0  R> R>  D+  R> R>  D+  ;

: T-  ( al am ah bl bm bh -- l m h )
  >R >R SWAP  >R SWAP >R
  0 SWAP 0  D-
  DUP  R> R>  D+  R> R>  D-  ;
```

Figure Three. Definitions for `T+` and `T-` using standard Forth words.

With minimal alterations to the kernel, a Forth-level access to the carry can be acquired, which will greatly enhance programming efficiency for producing extended-precision math operators. Only six words need to be added to the kernel to do this. They all involve direct manipulation of the carry.

These words are:

`SETC` "set-c"
(--)
The carry is set to a value of one.

`CLRC` "Clear-c"
(--)
The carry is set to a value of zero.

`+c` "plus-c"
(n1 n2 -- n3)
Add n2 and the carry to n1, giving the sum n3. The resulting carry is preserved in the carry bit.

`-c` "minus-c"
(n1 n2 -- n3)
Subtract n2 and the carry from n1, giving the difference n3. The resulting borrow is preserved in the carry bit.

`2*c` "two-star-c"
(n1 -- n2)
n2 is n1 shifted left one bit, with the carry going into the LSB, and the MSB going into the carry (circular left shift).

`c2/` "c-two-slash"
(n1 -- n2)
n2 is n1 shifted right one bit, with the carry going into the MSB, and the LSB going into the carry (circular right shift).

With these six words, extended-precision math operators can now be easily created—in high-level Forth—that are intuitive to understand and fast in operation.

Modifying the Kernel

As stated previously, Forth doesn't keep track of the carry that is generated when you add or subtract cells. The definition of "carry" is meant to distinguish the value which is associated with the operation of the Forth words, not the operations associated with an inner interpreter. The "carry bit" is the location the system uses to

```
  SIZE ARRAY TEMP                      \ make TEMP array of SIZE cells
: N+  ( num1 num2 #cells -- num3 [#cells long] )
  CLRC  DUP >R                         \ clear carry, store #cells
          0 DO  TEMP I + ! LOOP        \ store num2 in TEMP
        R@ 0 DO  TEMP SIZE 2/ I + + ! LOOP \ store num1 in TEMP
    0 0 R> - DO  0 I - 1 -             \ compute array index val
                 TEMP OVER SIZE 2/ + + @  \ retrieve num1(i) cell
                 SWAP TEMP + @         \ retrieve num2(i) cell
                 +c  LOOP ;            \ -c for N-
```

Figure Four. Arbitrary-precision math operators.

```
: UDM+  ( ual uah ubl ubh -- l m h )          ( unsigned )
  CLRC  >R SWAP >R
  +c R> R> +c  0 0 +c  ;

: DM+  ( al ah bl bh -- l m h )               ( signed )
  CLRC  >R SWAP >R
  +c R>  DUP R@
  +c SWAP  0< R>  0< +c  ;

: T+UD  ( al am ah ubl ubh -- l m h )         ( b is unsigned )
  CLRC  >R SWAP >R SWAP >R
  +c R> R> R>  SWAP R>
  +c R>  0 +c  ;

: T-UD  ( al am ah ubl ubh -- l m h )         ( b is unsigned )
  SETC  >R SWAP >R SWAP >R
  -c R> R> R>  SWAP >R
  -c R>  0 -c  ;
```

Figure Five. Extended-precision, mixed-math operators.

```
: T2*  ( al am ah -- l m h )
  CLRC  >R >R
  2*c R>  2*c R>  2*c  ;
```

This is much faster than:

```
: T2*  ( al am ah -- l m h )
  3DUP T+  ;
```

Figure Six. Extended-precision shift operators.

```
: UT2/  ( al am ah -- l m h )              ( logical shift right )
  CLRC  c2/ >R  c2/ >R  c2/ R>  R>  ;

: T2/  ( al am ah -- l m h )               ( arithmetic shift right )
  DUP 2*c DROP                             ( sign bit in c )
  c2/ >R  c2/ >R  c2/ R> R>
```

Figure Seven. Right-shift extended-precision operators.

```
: T2N*  ( al am ah count -- l m h )
  CLRC
    0 DO  >R >R 2*c R> 2*c R> 2*c
    LOOP  ;
```

or:

```
: T2N*  ( t # -- t )
  CLRC 1 -
    FOR >R >R 2*c  R> 2*c  R> 2*c
    NEXT  ;
```

Figure Eight. Multiple-shift, extended-precision operators.

Statement of Ownership, Management and Circulation

1) Title of Publication: Forth Dimensions
   Publication Number: U.S.P.S. 002-191
2) Date of Filing: 9/5/90
3) Frequency of Issue: Bi-Monthly
   No. of issues published annually: 6
   Annual subscription price: $24/36
4) Location of known office of publication: 1330 S Bascom Ave., Suite D, San Jose, Santa Clara County, California 95128-4502
5) Location of the headquarters or general business offices of the publisher: Same as above
6) Publisher: Forth Interest Group, P.O. Box 8231, San Jose, California, 95155
   Editor: Marlin Ouverson, Same as above
   Business Manager: Georgiana F. Shepherd, Same as above
7) Owner: Forth Interest Group, Same as above
8) Known bondholders, mortgagees, and other security holders owning or holding
   1% or more total amount of bonds, mortages and other securities: none
9) The purpose, function and non-profit status of this organization and the exempt status for Federal Income Tax purposes have not changed during the preceding 12 months.
10) Extent and nature of circulation

	Average No. copies/issue during preceding 12 mos.	Actual No. Copies of single issue nearest to filing date
A. Total no. copies printed:	2350	2200
B. Paid/requested circulation:		
1. Sales:	8	10
2. Mail subscription:	1880	1659
C. Total paid/requested circulation:	1888	1669
D. Free distribution by mail, carrier or other means: samples, complimentary and other free copies:	36	91
E. Total distribution:	1924	1760
F. Copies not distributed:		
1. Office use, left over, unaccounted, spoiled after printing:	426	440
2. Return form news agents:	0	0
G. TOTAL:	2350	2200

11) I certify that the statements made by me above are correct and complete
/s/ Georgiana F. Shepherd

store the carry, and isn't necessarily the location defined in the ALU (arithmetic logic unit) operation of machine-level instructions.

Therefore, you must create the mechanism to store the carry that is created by the machine's ALU when the Forth words + and - are used. In the ideal world, the definition of + and - would be altered to provide for the ALU-generated carry to be stored in a system variable I will name CARRY. This can easily be obtained from the machine's status register or directly from the ALU instructions themselves.

However, in order not to "bust" existing kernels and the words + and -, an alternative structure is preferable. First, create a variable or memory (register) location that will be used for storing the machine-level carry. Then write the carry-manipulation words in assembly, utilizing the CARRY memory location to preserve the machine-level operation carry.

This is necessary because the inner interpreter for non-Forth engines will most likely utilize ALU instructions which change the ALU carry value. Therefore, the state of the carry must be preserved before exiting those Forth words. On Forth engines such as the RTX 20xx family and the APL chip (SC32) the inner interpreter is hardware, so the carry is preserved in a hardware register and can be directly accessed by high-level Forth words.

Extended-Precision Math Made Easy

Now that I've explained how to gain access to the carry, let's use these new words to produce extended-precision math operators.

Normally, D+ and D- are defined in machine language. With the addition of these new words, they can now be defined as in Figure One; that was easy enough. For something harder, see Figure Two. Next, let's see how we would produce T+ and T- using standard Forth words (Figure Three).

Although both versions of these words take comparable code space, the words in Figure Two should run faster than those in Figure Three (D+ and D- should be slower than +c and -c). The words in Figure Two also have an intuitive symmetry that isn't present in Figure Three. After stack thrashing, the words in Figure Two simply perform a(l,m,h) +/- b(l,m,h) with carry/bor-

(Continued on page 40.)

Twelfth Annual

FORML CONFERENCE

The original technical conference
for professional Forth programmers, managers, vendors, and users.

Following Thanksgiving, November 23–25, 1990

Asilomar Conference Center
Monterey Peninsula overlooking the Pacific Ocean
Pacific Grove, California U.S.A.

Conference Theme: Forth in Industry

Papers are invited that address relevant issues in the development and use of Forth in industry. Papers about other Forth topics are also welcome.

Mail abstract(s) of approximately 100 words by October 1, 1990 to FORML, P.O. Box 8231, San Jose, CA 95155.

Completed papers are due November 1, 1990.

Conference Registration

Registration fee for conference attendees includes conference registration, coffee breaks, and notebook of papers submitted, and for everyone rooms Friday and Saturday, all meals including lunch Friday through lunch Sunday, wine and cheese parties Friday and Saturday nights, and use of Asilomar facilities.

Conference attendee in double room—$285 • Non-conference guest in same room—$160 • Children under 17 in same room—$120 • Infants under 2 years old in same room—free • Conference attendee in single room—$360

Register by calling the Forth Interest Group business office at (408) 277-0668 or writing to: **FORML Conference, Forth Interest Group, P.O. Box 8231, San Jose, CA 95155.**

About Asilomar

The Asilomar Conference Center combines excellent meeting and comfortable living accommodations. It is situated on the tip of the Monterey Peninsula overlooking the Pacific Ocean. Asilomar is part of the California State Park system; it occupies 105 secluded acres of forest and dune. If you like, you may jog on the beach before breakfast, join an informal discussion under a cypress tree after lunch, and exchange stories in front of a fireplace at the nightly wine and cheese parties. Guests of conference attendees may enjoy sightseeing along the beautiful Big Sur coast, visiting the new Monterey Aquarium, or shopping in nearby Carmel.

F83

FORTH AND THE THREE-NUMBER PROBLEM

LEONARD MORGENSTERN - MORAGA, CALIFORNIA

Years ago, before spreadsheets were invented, I had a problem: enter numbers at the keyboard, and have the computer display an answer. To accomplish this, I wrote a Forth word X to perform the calculation. Now, I could have my result by typing X at the end of each line. This worked fine, except that I developed a hatred for the letter *X*, and wondered if I could somehow make the action occur automatically as soon as the return key was pressed. Since then, I have discovered several ways, which I present here. To simplify the discussion, the problem will be reduced to the following, which I call the "three-number problem." The user enters three numbers and presses the return key. The computer displays their sum.

A programmer working in BASIC would use a direct approach: accept data into a buffer and analyze it. An experienced Forth programmer would recognize that this is exactly the sequence employed by Forth's outer interpreter, which is the sequence that gets input and converts it into Forth actions, and would make the appropriate modifications. Few other languages allow such freedom to tinker with their innards.

To simplify the discussion, the problem will be reduced to the following: The user enters three numbers and presses the return key. The computer displays their sum. This is the "three-number problem."

Copying BASIC's Approach

BASIC would solve the three-number problem by using built-in features of that language. A named area in memory, called a *string*, is defined, and the following four-step sequence repeated:

1. Get keyboard input into the string.
2. Scan the string for separators, and divide it into segments.
3. Convert the string's segments to numbers.
4. Calculate the sum and display it.

A fairly direct translation of the BASIC method into Forth is presented in Screen Two and is documented in Figure One. GETINPUT stores keyboard input in the buffer WORKAREA. -LEADING and LEX find the segments. NUMBER? performs the conversion.

The Forth Approach

Although there are differences in detail, the mechanism used by the BASIC approach is the same as that used by Forth's outer interpreter, which gets and interprets keyboard input. It turns out to be easy to insert a new action in it. This possibility might not occur to a programmer familiar only with conventional computer languages.

We do not program in Forth, rather, we program Forth...

The outer interpreter is called QUIT, so named because, when it is executed, it clears the data and return stacks, and awaits keyboard input. In effect, Forth "quits" what it is doing and starts over. (A more descriptive name would be RESTART.) I still remember how surprised I was to learn that QUIT is a colon definition, which anyone can modify, cautiously. I had expected to find it in the nameless stuff at the bottom of Forth, hidden from any but the expert eye. All QUITs work in basically the same way, but the word is too implementation-dependent to be copied directly from one Forth to another. F83 defines it as follows:

```
: QUIT ( -- )
```
QUIT is a colon definition.

```
SP0 @ 'TIB !
BLK OFF  [COMPILE] [
```
Housekeeping. Reset TIB, set input to keyboard, and turn off compilation.

BEGIN
Start a BEGIN ... AGAIN structure, which is an endless loop. It is not possible to terminate it in a "normal" way. The program can escape by 1) executing BYE to exit from Forth altogether, 2) executing QUIT explicitly, which starts the loop over again, or 3) executing ABORT or one of its variants which execute QUIT implicitly.

```
RP0 @ RP!
```
More housekeeping. Reset return stack.

STATUS
A deferred word, created by the defining word DEFER. Such a word has a slight resemblance to a variable, but differs in two respects: 1) It always contains the CFA of another Forth word, and 2) it executes that word instead of putting an address on the stack. To change the contents of a deferred word, use TO or IS, depending on your particular Forth. The default action of STATUS is CR, which starts a new line. (Try typing ' NOOP IS STATUS to see what happens.) The name implies that it should be used to show the status of the system—for example, by displaying certain variables—but it is far more versatile than

that. In Screen Three, it adds the three top numbers on the stack and exhibits the sum.

QUERY
Get 80 characters from the keyboard into TIB (Terminal Input Buffer).

RUN
Analyze the contents of TIB and execute each word in turn. F83 uses RUN instead of the standard INTERPRET, because F83's version of the latter does not permit multi-line compilation.

```
STATE @ NOT IF
." ok" THEN
```

STATE is a variable that contains FALSE if interpreting (for example, when executing keyboard input) and TRUE if compiling (when in a colon definition). If the former, issue the "ok" prompt. In the three-number problem, the prompt is an annoyance. To suppress it, you could patch the letters *ok* to spaces or something equally unobtrusive; but it is easier to write a new QUIT, as we will do in Screen Four. In some Forths, the "ok" prompt is deferred, and if you ever metacompile Forth, you should incorporate this feature. It would add a lot of flexibility—for example, the prompt might display something more informative than "ok," or you could solve the three-number problem without rewriting the QUIT loop. (I leave this as an exercise.)

AGAIN ;
End the loop and start over.

F83 has incorporated a deferred word, STATUS, into QUIT, providing an easy but limited way to make a change by simply assigning a new action. This is done in Screen Three, where ADD3 is the summation command. It checks the depth of the stack against a variable named DEPTH*. If there are three numbers present, they are added. If not, an error is signalled. The process is started by executing ADD'EM, which sets DEPTH* and modifies STATUS. RESTORE reverts to normal.

Writing a New QUIT Loop

Up to now, we have modified QUIT in a very limited way, by tinkering with STATUS. In Screen Four we will go all the way and write a new version of QUIT, named SUM'EM. It omits STATUS and the "ok" prompt. Compare it with QUIT, and note how the housekeeping has been meticulously copied. Also note its minute size, 51 bytes. Writing your own interpreter is not only easy, it is cheap!

To start summing, simply type SUM'EM. To exit, type QUIT. An error, such as an illegal Forth word or insufficient stack depth, will also exit. While SUM'EM is executing, input is interpreted actively, not passively. Any Forth word entered at the console is executed in a normal way. For example, if we type 3 3 * 4 5 <cr>, SUM'EM would understand the * as a command to multiply, and 18 would be displayed. By contrast, BASIC-style action would regard the * as an error, since it is not a number.

Although setting up a new outer interpreter is not terribly complicated, keep in mind that QUIT is the heart of Forth, and changing it is cardiac surgery. Do it carefully! Any mistake, and your system will hang up—if you are lucky. In particular, do not touch the housekeeping. Altering that is equivalent to writing your own Forth.

Forth vs. Traditional Computer Languages

This example teaches several lessons about computer languages in general and Forth in particular. BASIC is a "traditional" language, a class that includes Pascal, Fortran, and C. It has three levels: assembler, language, and application, all sharply separated from one another. We say that an application is "written" in the language. Only an expert can change the language itself, according to the tacit principle that providing access to the computer is only the secondary purpose of a language; the primary is protecting the computer from the user.

By contrast, these levels do not exist in Forth, and you can do as much damage as you wish. CODE words are often called "low-level" and colon definitions "high-level," but the fact is that, once defined, they are all at the same level. A CODE word and a colon definition with the same action are indistinguishable without decompiling them, except that CODE definitions usually run faster. Furthermore, we do not program *in* Forth, rather, we program *Forth*, by modifying and extending it to suit our needs. For expert users, we might create a large number of powerful, specialized words, and for ordinary users a small English-like subset.

Conventional-language programmers find it hard to visualize such a language. To them, it is just another evidence of Forth's "weirdness." Two incidents from my own experience illustrate the problem. I was explaining to a relative, a man with a giant I.Q. and a lot of programming experience, how flexible Forth is. The conversation went like this:

Leonard: In Forth, you could redefine + to subtract instead of add.
HW: You can do that in Pascal. Just define a function.
Leonard: (After vain attempts to explain the difference) If I write *a:= 3 + 4* what is the value of *a*?
HW: Seven.
Leonard: No, it's -1. Remember, we changed + to -.

The following exchange occurred at an Asilomar conference:

Speaker #1: Forth is not a computer language. It is a way to program a computer. BASIC is a computer language.
Speaker #2: No, Forth *is* a computer language, and BASIC is not.

I think the second was right. Forth is like a natural language, such as English. Conventional computer languages are more like cable code, full of arbitrary phrases such as *"AZCEG Your message received."*

Summary

Problems are sometimes solved in Forth at deeper levels than is possible in a conventional languages such as BASIC.

Leonard Morgenstern is a retired pathologist and computer hobbyist. His interest in Forth goes back over ten years. Currently, he is a sysop of the Forth RoundTable on the GEnie network.

```
Screen 2
\ Add figures BASIC-style            NLM900525
: -LEADING ( a l -- a' l' )
BL SKIP ;
: GETINPUT ( a l -- )
OVER 1+ SWAP EXPECT   SPAN @ SWAP  2DUP C!
+ 1+ BL SWAP C! ;
: LEX ( a l b -- a[tail] l[tail] a[head] l[head] )
>R 2DUP R> SCAN   DUP >R   2SWAP R> - ;
CREATE WORKAREA 82 ALLOT
: ADD'EM ( -- )
  WORKAREA DUP 79 GETINPUT COUNT
          BEGIN  -LEADING  DUP 0<>
          WHILE  BL  LEX DROP 1- NUMBER?
          NOT IF BEEP THEN  2SWAP
          REPEAT 2DROP D+ D+ 3 SPACES D.  ;
\ 261 bytes



Screen 3
\Add figures using STATUS            NLM900525
VARIABLE DEPTH*
: ADD3  DEPTH DEPTH* @ - 3 >= IF 2 SPACES + + .THEN CR ;
: RESTORE   ['] CR IS STATUS ;
: ADD'EM ['] ADD3 IS STATUS  DEPTH DEPTH* ! ;
\ 109 bytes



Screen 4
\ Add figures with QUIT loop         NLM900525
: SUM'EM  SP0 @ 'TIB ! BLK OFF [COMPILE] [
  BEGIN RP0 @ RP! QUERY RUN
  + + .  CR AGAIN ;
\ 51 bytes
```

(Figure on page 31.)

FIG
MAIL ORDER FORM

MEMBERSHIP IN THE FORTH INTEREST GROUP

112 - MEMBERSHIP in the FORTH INTEREST GROUP and Volume 12 (May/June '90 - March/April '91) of *Forth Dimensions.* No sales tax, handling fee, or discount on membership. See back page of this order form.

The Forth Interest Group is a world-wide, non-profit, member-supported organization with over 2,000 members and 55 chapters. FIG membership includes a subscription to the bi-monthly magazine, *Forth Dimensions.* FIG also offers its members an on-line data base, a large selection of Forth literature and many other services. Cost is $30.00 per year for USA & Canada surface mail; $36.00 Canada air mail; all other countries $42.00 per year. Annual membership dues are based on the membership year, from May 1 to April 30.

When you join, you will receive issues that have already been circulated for the current volume of *Forth Dimensions*, and subsequent issues will be mailed to you as they are published. You will also receive a membership card and number which entitles you to a 10% discount on publications from FIG. Your member number will be required to receive the discount, so keep it handy.

Dues are not deductable as a charitable contribution for U.S. federal income tax purposes, but may be deductable as a business expense where applicable.

HOW TO USE THIS FORM

Most items list three different price catagories:

USA, Canada and Mexico / Other countries Surface Mail / Other countries Air Mail

Note: Where only two prices are listed, Surface Mail is not available.

Please enter your order on the back page of this form and send with your payment to the Forth Interest Group.

FORTH DIMENSIONS BACK VOLUMES

The six issues of the volume year (May-April)

101 -Vol.1	FORTH Dimensions (1979/80)	$15/16/18_____
102 -Vol.2	FORTH Dimensions (1980/81)	$15/16/18_____
103 -Vol.3	FORTH Dimensions (1981/82)	$15/16/18_____
104 -Vol.4	FORTH Dimensions (1982/83)	$15/16/18_____
105 -Vol.5	FORTH Dimensions (1983/84)	$15/16/18_____
106 -Vol.6	FORTH Dimensions (1984/85)	$15/16/18_____
107 -Vol.7	FORTH Dimensions (1985/86)	$20/22/25_____
108 -Vol.8	FORTH Dimensions (1986/87)	$20/22/25_____
109 -Vol.9	FORTH Dimensions (1987/88)	$20/22/25_____
110 -Vol.10	FORTH Dimensions (1988/89)	$20/22/25_____
111 -Vol.11	FORTH Dimensions (1989/90)	$20/22/25_____

FORML CONFERENCE PROCEEDINGS

FORML (Forth Modification Laboratory) is an informal forum for sharing and discussing new or unproven proposals intended to benefit Forth. Proceedings are a compilation of papers and abstracts presented at the annual conference. FORML is part of the Forth Interest Group.

310 - FORML PROCEEDINGS 1980 $30/31/40_____
Technical papers on the Forth language and extensions.

311 - FORML PROCEEDINGS 1981 $45/48/55_____
Nucleus, interactive, and extensible layers; metacompilation, system development, file systems, other languages and operating systems; applications.

312 - FORML PROCEEDINGS 1982 $30/31/40_____
Forth machine topics, implementation topics, vectored execution, system development, file systems and languages, applications.

313 - FORML PROCEEDINGS 1983 $30/32/40_____
Forth in hardware, Forth implementations, future stratgy, arithmetic and floating point, file systems, coding conventions, functional programming.

314 - FORML PROCEEDINGS 1984 $30/33/40_____
Expert Forth systems, philosophy, implementing Forth systems, new Forth directions, interfacing Forth to operating systems, adding local variables.

316 - FORML PROCEEDINGS 1986 $30/32/40_____
Forth internals, Methods, Standards, Forth processors, Artificial Intelligence, Applications.

317 - FORML PROCEEDINGS 1987 $40/43/50_____
Includes papers from '87 euroFORML Conference. 32 bit FORTH, neural networks, control structures, AI, optimizing compilers, hypertext and more.

318 - FORML PROCEEDINGS 1988 $24/25/34_____
Human interfaces, Simple Robotics Kernel System, MODUL Forth, Language topics, hardware, Wil's workings & Ting's philosophy, Forth hardware applications, ANS Forth session, Future of Forth in AI Applications.

380 - AUSTRALIAN PROCEEDINGS 1988 $24/25/34_____
Proceedings from the first Australian Forth Symposium held May, 1988 at the University of Technology in Sydney. Subjects include training, parallel processing, programmable controllers, Prolog in Forth, simulations & applications.

319 - FORML PROCEEDINGS 1989 $40/43/50_____
Includes papers from '89 euroFORML. PASCAL to Forth, extensible optimizer for compiling, 3-D measurement using object-oriented Forth, CRC polynomials, Harris C cross-compiler, modular approach to robotic control, RTX recompiler for on-line maintenance, module, trainable neural nets.

BOOKS ABOUT FORTH

200 - ALL ABOUT FORTH, 2nd ed., March 1983 $28/29/38_____
Glen B. Haydon
An annotated glossary for MVP Forth: a 79-Standard Forth.

201 - ALL ABOUT FORTH, 3rd ed., June 1990 $90/92/105_____
Glen B. Haydon
An annotated glossary of most Forth words in common usage, including F-79, F-83, F-PC, MVP-Forth. Implementation examples in high-level Forth and/or 8086/88 assembler. Useful commentary is given for each entry.

210 - THE COMPLETE FORTH $14/15/19_____
Alan Winfield
A comprehensive introduction including problems with answers (Forth79)

217 - F83 SOURCE $20/21/30_____
Henry Laxen & Michael Perry
A complete listing of F83 including source and shadow screens. Includes introduction on getting started.

218 - FOOTSTEPS IN AN EMPTY VALLEY $25/26/35_____
Dr.C.H. Ting
A thorough examination and explanation of the NC4000 Forth chip including the complete source to cmForth from Charles Moore.

219 - FORTH: A TEXT AND REFERENCE $28/29/38_____
Mahlon G. Kelly & Nicholas Spies
A textbook approach to Forth with comprehensive references to MMS-FORTH and the 79 and 83 Forth Standards.

220 - FORTH ENCYCLOPEDIA $30/32/40_____
Mitch Derick & Linda Baker
A detailed look at each fig-FORTH instruction.

232 - FORTH NOTEBOOK $25/26/35_____
Dr.C.H.Ting
Good examples and applications. Great learning aid. PolyFORTH is the dialect used. Some conversion advice is included. Code is well documented.

232a - FORTH NOTEBOOK II $25/26/35_____
Dr. C. H. Ting
Collection of research papers on various topics as image processing, parallel processing and miscellaneous applications.
235 - INSIDE F-83 $25/26/35_____
Dr.C.H.Ting
Invaluable for those using F-83.
237 - LIBRARY OF FORTH ROUTINES AND UTILITIES
James D. Terry $23/25/35_____
Comprehensive collection of professional quality computer code for Forth; offers routines that can be put to use in almost any Forth application, including expert systems and natural language interfaces.
240 - MASTERING FORTH, 2nd Edition $22/23/28_____
Anita Anderson & Martin Tracy
A step-by-step tutorial including each of the commands of the Forth-83 International Standard; with utilities, extensions and numerous examples.
242 - OBJECT ORIENTED FORTH $25/26/30_____
Dick Pountain
Implementation of Data Structures. First book to make object orientated programming available to users of even very small home computers.
244 - STACK COMPUTERS, THE NEW WAVE $62/65/72_____
Philip J. Koopman, Jr.
Presents an alternative to Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC) by showing the strengths and weaknesses of stack machines. (hard cover only)
245 - STARTING FORTH, 2nd Edition (soft cover)
Leo Brodie $29/30/38_____
In this edition of Starting Forth, the most popular and complete introduction to Forth, syntax has been expanded to include the new Forth '83 Standard.
267 - TOOLBOOK OF FORTH $23/25/35_____
(Dr.Dobb's) Edited by Marlin Ouverson
Expanded and revised versions of the best Forth articles collected in the pages of Dr.Dobb's Journal.
267a - TOOLBOOK, V.1 & DISK (MS-DOS) $40/42/50_____
268 - TOOLBOOK OF FORTH, V.2 $30/32/40_____
(Dr. Dobbs)
Complete anthology of FORTH programminmg techniques and developments, picks up where V.1 left off. Topics include programming windows, extended control structures, design of a FORTH target compiler and more.
268a - TOOLBOOK, V.2 & DISK (MS-DOS) $46/48/56_____
268b - TOOLBOOK, V.2 & DISK (MAC) $46/48/56_____

DR. DOBB'S JOURNAL
Annual Forth issue, includes code for various Forth applications.

422 - Sept, 1982 $5/6/7_____
423 - Sept, 1983
424 - Sept, 1984

SILICON COMPOSERS PRODUCT MANUALS
1110 - SC/FOX PCS USER MANUAL $35/38/45_____
Silicon Composers' plug-in RTX2000 parallel coporcessor for PC/AT/386. Manual includes operation; hardware & PC interface description; RTX2000 architecture and instruction set; & FCompiler optimizing Forth compiler glossaries for the RTX2000. SC/Forth is also available for this board.
1112 - SC/FORTH USER MANUAL $35/38/45_____
Users manual for Silicon Composers Forth, a time-sliced multitasking 83-Standard Forth with exceptions for the RTX2000 chip. Manual includes complete word set glossary; description of SC/Forth internals; multitasking and turnkey application examples; and block/text file support options.

ACM - SIGFORTH
The ACM SIGForth Newsletter is published quarterly by the Association of Computing Machinery, Inc. SIGForth's focus is on the development & refinement of concepts, methods and techniques needed by Forth professionals.
901 - Volume 1, Summer 1989 $6/7/9_____
Metacompiler in cmForth, Forth exception handler, string case state-ment for UF/Forth.
902 - Volume 1#3, Fall 1989 $6/7/9_____
RCA 1802 software simulator, tutorial on multiple threaded vocabularies.
903 - Volume 1#4, Winter 1989 $6/7/9_____
Stack frames in Forth, duals: an alternative to variables, PocketForth.
931 - SIGForth Workshop Proceedings 1989 $20/21/26_____
(Software Engineering)
Multi-tasking, interrupt driven systems, object oriented Forth, error recovery & control, virtual memory support, signal processing.

ROCHESTER PROCEEDINGS
The Institute for Applied Forth Research, Inc. is a non-profit organization which supports and promotes the application of Forth. It sponsors the annual Rochester Forth Conference.
321 - ROCHESTER 1981 $25/28/35_____
(Standards Conference)
322 - ROCHESTER 1982 $25/28/35_____
(Data bases & Process Control)
323 - ROCHESTER 1983 $25/28/35_____
(Forth Applications)
325 - ROCHESTER 1985 $20/21/25_____
(Software Management & Engineering)
Improving software productivity, using Forth in a space shuttle experiment, automation of an airport, development of MAGIC/L and a Forth based business applications language.
326 - ROCHESTER 1986 $20/21/25_____
(Real-Time Artificial Intellegence)
Object-oriented programming, software tools, implementations, space applications, process monitoring, HHC, and mathematics.
327 - ROCHESTER 1987 $20/21/25_____
(Comparative Computer Architectures)
Multi-Stack Forth machines, parallel processing, biological computing, and VLSI implementations.
328 - ROCHESTER 1988 $25/26/33_____
(Programming Environments)
Operating systems and languages, UNIX, TICOL, X-Script, RPL (Reverse Polish Lisp), RTX, WISC, NC4000.
329 - ROCHESTER 1989 $25/26/33_____
(Industrial Automation)
Embedded systems, object oriented programming, 32bit SC32 and Harris RTX 2000 and 4000 processors, image processing.

JOURNAL OF FORTH APPLICATION & RESEARCH
Refereed technical journal published by the Inst. for Applied Forth Research.
401 - Volume 1, Robotics/Data Structures $30/31/38_____
404 - Volume 2#2, Real-Time Systems $15/16/18_____
405 - Volume 2#3, Enhancing Forth $15/16/18_____
406 - Volume 2#4, Extended Addressing $15/16/18_____
407 - Volume 3#1, Forth-based Lab Systems $15/16/18_____
409 - Volume 3#3, Application Languages $15/16/18_____
410 - Volume 3#4, Applications, Arithmatic $15/16/18_____
411 - Volume 4#1, Expert Systems in Forth $15/16/18_____
412 - Volume 4#3, REPTIL $15/16/18_____
413 - Volume 4#4, Embedding Languages in Forth $15/16/18_____
414 - Volume 5#2, Mathematics, ANS Standard $15/16/18_____ NEW
415 - Volume 5#3, From Russia with Forth $15/16/18_____
416 - Volume 5#4, Forth Processors, Parallel Forth $15/16/18_____
417 - Volume 6#1, Harris RTX2000, Scientific Prog. $15/16/18_____

Odds & Ends
D100 - Forth Fundamentals V2 $10/11/15_____
C. Kevin McCabe
Language Glossary, fig-Forth and Forth 79.
D200 - Starting Forth $23/25/32_____
Leo Brodie
First edition, 1981 (hard cover)

MISCELLANEOUS
601 - T-SHIRT SIZE_____Small, Medium, Large, Extra-Large
"May the Forth Be With You"
White design on a dark blue shirt. $12/13/15_____
602 - POSTER (BYTE Cover) $5/6/7 _____
683 - FORTH-83 HANDY REFERENCE CARD FREE _____

FIG DISK LIBRARY

"Contributions from the Forth Community" are author submitted donations, generally source, for a variety of computers on their respective disk formats. The cost is $6/9 per disk "(1)" or $25/28 for five. The files usage is determined by the author as Public Domain, Shareware, or use with some restrictions. This library does not contain "For Sale" applications.

To submit your "Contributions", request a Submission form from the FIG office.

C001 (1) - FLOAT4th.BLK V1.4 Robert L. Smith $6/9_____
Software Floating-Point for fig, Poly, 79-STD, 83-STD Forths. IEEE Short 32-bit, Four standard functions, Square Root and Log. **IBM**.

C100 (1) - F83 V2.01, Mike Perry & Henry Laxen $6/9_____
The newest version that has been ported to a variety of machines. Editor, assembler, decompiler, meta-compiler. Source and shadow screens. Manual available separately. Base for other F83 applications. **IBM, 83.**

C200 (5) - F-PC V3.53, Tom Zimmer $25/28_____
A full Forth system with pull-down menus, sequential files, editor forward assembler, meta-compiler, floating point. Complete source and Help files. Manual for V3.5 available separately (see items 350 & 351 below). Base for other F-PC applications. Hard disk recommended. **IBM, 83.**

C201a (2) - F-PC TEACH, Lessons 0-5 $12/15_____
V2.25, J. Brown
Forth classroom on disk. First five lessons from Jack Brown of BC Institute of Technology on learning Forth. **IBM**, F-PC.

C202 (1) - VP-Planner Floating Point for F-PC $6/9_____
V1.01, Jack Brown
Software floating point engine behind the VP-Planner spreadsheet. 80-bit (temporary-real) routines with Transcendental Functions, NUMBER I/O support, vectors to support numeric coprosessor overlay and user NAN checking. **IBM**, F-PC.

NEW **C203a** (3) - F-PC Graphics V4.2f, Mark Smiley $18/21_____
The latest versions of a number of new graphics routines including CGA, EGA and VGA suppport, with numerous improvements over earlier versions created or supported by Mark Smiley. IBM, F-PC.

C300 - PocketForth: V1.4, Chris Heilman $6/9_____
Smallest complete Forth for the Mac. Access to all Mac functions, files, graphics, floating point, macros, create stand-alone applications and DA's. Source and manual included. **MAC**, based on fig & *Starting Forth*

C401 (1) - JLISP V1.0, Nick Didkovsky $6/9_____
LISP interpreter invoked from Amiga JForth. The nucleus of the interpreter is the result of Martin Tracy's work. It has been extended to allow the LISP interpreter to link to and execute JForth words. It can communicate with JForth's ODE (Object Development Environment). **AMIGA,** 83.

REFERENCE

306 - ANS X3J14 BASIS DOCUMENT $15/16/20_____
Current - August 1990
Working document of the X3J14 ANS Forth Committee, changes frequently, but useful as a working tool.

305 - FORTH 83-STANDARD $15/16/18_____
Authoritative description of 83-Standard Forth. Reference, not instruction.

300 - FORTH 79-STANDARD $15/16/18_____
The authoritative description of 79-Standard Forth. Of historical interest.

305 - SYSTEMS GUIDE TO fig-FORTH $25/28/30_____
C. H. Ting - 2nd Edition, 1989
Hows and Whys of the fig-Forth Model by Bill Ragsdale, Internal structure of fig-Forth system.

340 - BIBLIOGRAPHY OF FORTH REFERENCES
3rd Edition, January 1987 $18/19/25_____
Over 1900 references to Forth articles throughout computer literature.

350 - F-PC USERS MANUAL, 2nd Edition, V3.5 $20/21/27_____
Users Manual to the public domain Forth system optimized for the IBM-PC/XT/AT computer. A fat, fast system with many tools.

351 - F-PC TECHNICAL REFERENCE MANUAL $30/32/40_____
A must if you need to know the inner workings of F-PC.

MORE ON FORTH ENGINES SERIES (MORE ON NC4000) EDITOR, C.H. TING

$15/16/18_____

801 - VOLUME 1, JULY 1986 - FIG-Tree style forum on NC4000. Topics including bugs, products, tips, benchmarks and and Chuck Moore's teleconference.

802 - VOLUME 2, OCTOBER 1986 - NC4000 User's Group's Newsletter. Hardware enhancements, software and many utility programs.

803 - VOLUME 3, JANUARY 1987 - NC6000/5000 advanced info., QUAN, DEPTH, and CASE, DROP, PICK, floating point, A/D converters.

804 - VOLUME 4, APRIL 1987 - Chuck Moore's AppNotes 1-7, line drawing, screen editor, 32 bit Forth engine design. Fourier transform, Tiny Modula-2.

805 - VOLUME 5, JULY 1987 - Moore's AppNotes 8-15, Harris FORCE Toolbox, Fk3 parallel processing, high speed pattern generator & 32 bit Forth simulator.

806 - VOLUME 6, NOVEMBER 1987 - mpFORTH, MIDI interface, multiplier-accumulator, Yin-Yang, NC4000 decompiler, Harris FORCE and MISC M17.

807 - VOLUME 7, MARCH 1988 - AT/Force technical notes, fixes for interrupt, interface to Intel chips, cross compiler, decompiler and debugger.

808 - VOLUME 8, MAY 1988 - Eight application notes from Novix on TI switch, disk controller, PBX laser printer, memory control and interrupts.

809 - VOLUME 9, NOVEMBER 1988 - Application notes on F68HC11, Super 8, LISP kernel and parallel computing on the NC4016.

810 - VOLUME 10, JANUARY 1989 - RTX reprints from 1988 Rochester Forth Conference, object oriented cmForth, lesser Forth engines.

811 - VOLUME 11, JULY 1989 - RTX Supplement to *Footsteps in an Empty Valley*, SC32, 32 bit Forth engine, RTX interrupts/utility.

812 - VOLUME 12, APRIL 1990 - ShBoom Chip architecture and instructions, Neural Computing Module NCM3232, pigForth, Binary Radix Sort on 80286, 68010, and RTX2000.

FORTH MODEL LIBRARY

Model applications disks below are first releases of professionally developed Forth applications. 5 1/4" disks are IBM MS-DOS 2.0 and up compatible with Forth-83 systems listed below. Please specify disk size when ordering
Laxen/Perry F83 MasterFORTH 1.0 PolyFORTH (R) 11
LMI PC/FORTH 3.0 TaskFORTH 1.0
Macintosh 3 1/2" disks are available for MasterFORTH systems only.

701 - A FORTH LIST HANDLER V.1 $40/43_____
by Martin J. Tracy
Forth is extended with list primitives to provide a flexible high-speed environment for AI. ELISA and Winston & Horn's micro-LISP included as examples. Documentation is included on the disk.

702 - A FORTH SPREADSHEET V.2 $40/43_____
by Craig A. Lindley
This model spreadsheet first appeared in Forth Dimensions Volume 7, issues 1 and 2. Those issues contain the documentation for this disk.

703 - AUTOMATIC STRUCTURE CHARTS V.3 $40/43_____
by Kim Harris
These tools for the analysis of large Forth programs were first presented at the 1985 FORML conference. Documentation is in the 1985 FORML Proceedings.

704 - A SIMPLE INFERENCE ENGINE V.4 $40/43_____
by Martin J. Tracy
Based on the Inference Engine in Winston & Horn's book of LISP; takes you from pattern variables to complete unification algorithm. Accompanied throughout with running commentary on Forth philosophy and style.

706 - THE MATH BOX V.6 $40/43_____
by Nathaniel Grossman
Collection of mathematical routines by the foremost author on math in Forth. Extended double precision arithmetic, a complete 32-bit, fixed-point math package and auto-ranging text graphics. Includes utilities for rapid polynomial evaluation, continued fractions and Monte Carlo factorization.

ASSEMBLY LANGUAGE SOURCE CODE LISTINGS

Listings of fig-FORTH for specific CPUs and machines with compiler security and variable length names. $15/16/18_____

513 - 1802/March 81
514 - 6502/September 80
515 - 6800/May 79
516 - 6809/June 80
517 - 8080/September 79
518 - 8086/88/March 81
519 - 9900/March 81
521 - APPLE II/August 81
523 - IBM-PC/March 84
526 - PDP-11/January 80
527 - VAX/October 82
528 - Z80/September 82

HISTORICAL DOCUMENTS

502 - fig-FORTH INSTALLATION MANUAL $15/16/18_____
Glossary model editor - we recommend you purchase this manual when purchasing the source code listing.

502 - KITT PEAK PRIMER $25/26/34_____
One of the first institutional books on Forth. Of historical interest.

503 - USING FORTH, 2nd Edition, 1980 $15/16/18_____
FORTH, Inc.

FORTH INTEREST GROUP

P.O. BOX 8231 SAN JOSE, CALIFORNIA 95155 (408) 277-0668 (408)286-8988(FAX)

Name_____ Member#_____

Company_____

Street_____

City_____

State/Prov._____ Zip_____

Country_____

Daytime Phone_____

OFFICE USE ONLY

By_____ Date_____ Type_____

Shipped by_____ Date_____

UPS USPS XRDS

Wt._____ Amt._____

BO By_____ Date_____

Wt._____ Amt._____

ITEM #	TITLE	QTY.	UNIT PRICE	TOTAL
112	*MEMBERSHIP*			*SEE BELOW*

☐ CHECK ENCLOSED (Payable to: Forth Interest Group)

☐ VISA ☐ MasterCard

Card Number_____

Expiration Date_____

Signature_____
($15.00 minimum on all VISA/MasterCard orders)

SUB-TOTAL	
10% MEMBER DISCOUNT MEMBER NUMBER_____	
SUB-TOTAL	
CALIF. RESIDENTS ADD SALES TAX	
HANDLING FEE	$2.00
___NEW ___RENEWAL $30/36/42 MEMBERSHIP	
Enclosed is $30/36/42 for 1 full years dues. This includes $24/30/36 for Forth Dimensions.	

PAYMENT MUST ACCOMPANY ALL ORDERS

MAIL ORDERS
Send to:
Forth Interest Group
PO Box 8231
San Jose, CA 95155

PHONE ORDERS
Call 408/277-0668 to place credit card orders or for customer service. Hours: M-F, 9am - 5pm PST.

PRICES
All orders must be prepaid. Prices are subject to change without notice. Credit card orders will be sent and billed at current prices. $15 minimum on charge orders. Checks must be in US$, drawn on a US bank. A $10 charge will be added for returned checks.

POSTAGE & HANDLING
Prices include shipping. A $2.00 handling fee is required with all orders.

SHIPPING TIME
Books in stock are shipped within seven days of receipt of the order. Please allow 4-6 weeks for out-of-stock books (deliveries in most cases will be much sooner).

SALES TAX
Deliveries to Fresno, Los Angeles, Riverside, Sacramento, San Francisco, San Mateo, Inyo, San Benito, Imperial, Monterey, Santa Barbara, San Bernardino, and Santa Cruz counties, add 6.75%. Contra Costa, San Diego, Alameda, and Santa Clara counties, add 7.25%. Other California counties, add 6.25%.

FORTH ASSEMBLER & FREE USE OF LABELS

CHESTER H. PAGE - SILVER SPRING, MARYLAND

The 6502 family of processors has the very convenient conditional branching commands BEQ, BPL, etc. They are followed by a one-byte value of a jump to be taken, forward or backward, of up to 127 bytes. In a conventional assembler, an address label can be used; the assembler will convert the data to the proper jump distance. Labels are also used as destinations for JSR and JMP commands.

This assembler has a number of advantages.

I decided that I needed a Forth assembler with all these advantages. Entering DLAB <name> (DLAB for "define label") creates a label; the <name> is stored in a table (using a fixed number of characters), followed by the address held in the dictionary pointer when the DLAB command was entered. When a label address is needed, entering ULAB <name> (for "use label") stores the <name> in a separate table, followed by the address held by DP. At the end of the assembly procedure, END steps through the ULAB use-request table, reads each name, finds it in the defining table, and stores the required jump address at the required place or computes the jump distance and stores one byte.

This assembler has a number of advantages. Since carrying out the assembly—primarily definition of primitive words—is the execution of Forth operations, other Forth operations can be carried out, such as computing an address to be compiled. A good example is:

```
' NEXT 2+ JMP,
```

Another Forth operation that can be

```
ASSEMBLER SCR # 1
 0 \ Using DLAB and ULAB with labels                          24APR89CHP
 1 HEX
 2 VOCABULARY ASSEMBLER     ASSEMBLER DEFINITIONS
 3 VARIABLE MODE    VARIABLE MODE.KEY
 4 VARIABLE LONG.ADDR
 5 5 CONSTANT LAB.LEN
 6 8E00 CONSTANT LAB.TAB
 7 8900 CONSTANT PLACE.TAB
 8 VARIABLE LAB.PTR
 9 VARIABLE PLACE.PTR
10
11 : CLEAR.TABLES PLACE.TAB 800 0 FILL PLACE.TAB PLACE.PTR !
12     LAB.TAB LAB.PTR ! ;
13 : , HERE ! 2 ALLOT ;
14 : C, HERE C! 1 ALLOT ;
15 .( #1)   -->


ASSEMBLER SCR # 2
                                                             24APR89CHP
 0 \
 1 CREATE FIND.LAB PRIM 04A9 , E585 , 00B5 , E699 , E800 ,
 2 C4C8 , D0E5 , A0F5 , B100 , D0E6 , C004 , F000 , 511C ,
 3 D0E8 , C807 , 00C0 , EF90 , 11F0 , A518 , 69E6 , 8500 , A5E6 ,
 4 69E7 , 8500 , A0E7 , F000 , CADC , CACA , C0CA , D000 ,
 5 980B , 0095 , 0195 , 0295 , 0395 , 13F0 , 00A0 , E6B1 , 0295 ,
 6 E8B1 , 0095 , B1C8 , 95E6 , B103 , 95E8 , 4C01 , ' NEXT 2+ ,
 7 LAB.LEN ' FIND.LAB OVER OVER 21 + C! OVER 2+ OVER 2A + C!
 8   4B + C!
 9 : ULAB BL WORD DUP 1+ SWAP C@ PLACE.PTR @ SWAP CMOVE LAB.LEN
10   PLACE.PTR +! HERE PLACE.PTR @ ! 2 PLACE.PTR +! ;
11 : ?USED LAB.PTR @ LAB.TAB FIND.LAB = 0= IF ." Label " LAB.PTR
12 @ 5 TYPE ."  already exists" CLEAR.TABLES SP! QUIT THEN ;
13 : DLAB BL WORD DUP 1+ SWAP C@ LAB.PTR @ SWAP CMOVE ?USED
14 LAB.LEN LAB.PTR +! HERE LAB.PTR @ ! 2 LAB.PTR +! ;
15 .( #2)   -->


ASSEMBLER SCR # 3
                                                             24APR89CHP
 0 \  Modes
 1 : ZP 0 MODE ! 0 MODE.KEY ! ;     \ Adds 4 to opcode
 2       \ ZP is default mode
 3 : ,X 1 MODE ! 1 MODE.KEY ! ;     \ Adds 14  (zero page,X)
 4 : ,Y 2 MODE ! 202 MODE.KEY ! ;   \ Adds 14 - LDX, STX,  only
 5 : X) 3 MODE ! 4 MODE.KEY ! ;     \ Adds 0    (ZP,X)
 6 : )Y 4 MODE ! 8 MODE.KEY ! ;     \ Adds 10   (ZP),Y
 7 : #  5 MODE ! 110 MODE.KEY ! ;   \ Adds 8    Immediate
 8 : ,A 6 MODE ! 20 MODE.KEY ! ;    \ Adds 8    Accumulator
 9 : )  7 MODE ! 40 MODE.KEY ! ;    \ Adds 2C - Indirect JMPs only
10 \    8                            Adds C  - Absolute address
11 \    9                            Adds 1C - Absolute,X
12 \    A                            Adds 18 - Absolute,Y
13 CREATE ADD.TABLE     \ Indexed by mode value
14   1404 , 0014 , 0810 , 2C08 , 1C0C , 18 C,
15 .( #3)   -->
```

inserted between definitions of primitive words is the definition of a colon word. For these reasons, I use ASSEMBLE instead of CODE at the beginning of the definition of a primitive, and no END-CODE. A single END at the conclusion of the need for labels takes care of everything.

The maximum label length can be specified (I use five characters), and all extra characters are ignored. In the application which stimulated the design of this assembler, I allocated $8900–8DFF for the use-table and 8E00–90FF for the define-table. At five characters per label and two bytes for each address, this allowed for 219 label uses and for the definition of 73 labels.

Since JMP and JSR need two-byte addresses, if the stack is empty when one of these commands is encountered (i.e., when preceded by a label instead of an explicit address), a 0000 target address is automatically compiled. Since the conditional branch instructions are always assumed to be preceded by a target label, a dummy jump length of $FF is compiled. When END tries to store a defined address at a requested location, it first reads a dummy byte, which will be either 00 or FF. If 00, the address is stored, overwriting the 0000 dummy; if FF is found, the jump distance is computed and the dummy FF is overwritten by a single byte. If the computed distance is more than 127 bytes an error message is given, stating that "Branch from <name> at address [nnnn] to address [mmmm] is too far." This detailed information helps to locate the program error.

A Forth word to function like the BASIC word CALL (to JSR a machine code routine at the address specified on the parameter stack) can be tricky. It requires an internal JSR request, with the JSR address to be changed before CALL is used.

The following technique will work:

```
ASSEMBLE CALL 0 ,X LDA, ULAB
CALL1 FF00 STA, 1 ,X LDA, ULAB
CALL2 FF00 STA, JSR, -2 ALLOT
DLAB CALL1 1 ALLOT DLAB CALL2
1 ALLOT INX, INX, ' NEXT 2+
JMP,
```

Assembling this sets up the label tables and compiles:

```
LDA 0,X
STA FF00
LDA 1,X
```

```
ASSEMBLER SCR # 4
 0 \ A is a given address                                    24APR89CHP
 1 \ C is address returned by opcode mnemonic
 2 : ?LEGAL ( C---C) DUP 1+ C@ MODE.KEY @ AND FF AND
 3  ABORT"  Illegal Opcode" DUP C@ 20 =        \ Check for ,A
 4    0= IF OVER 100 U< 0= IF MODE.KEY @ 0C AND
 5    ABORT"  Illegal Indirect" THEN THEN ;
 6
 7 : ABS.ADDR DUP 1+ @ MODE.KEY @ DUP 3C AND
 8 ABORT" Illegal address" DUP 40 = IF DROP DROP ELSE AND 200 =
 9    IF -1 MODE +! THEN  8 MODE +! THEN 1 LONG.ADDR ! ;
10 : ?ZP ( C---C) MODE.KEY @ 20 = 0= IF OVER 100 U< 0=
11  OVER 1+ C@ MODE.KEY @ OVER OR 262 = SWAP 3F = OR OR
12   IF ABS.ADDR THEN THEN ;
13 : ?IMM DUP 1+ @ MODE.KEY @ AND 100 = IF -2 MODE +! THEN ;
14   .( #4)  -->
15 \ Special treatment of immediate with CPX, CPY, STX, or STY,


ASSEMBLER SCR # 5
 0 \ END reports DLAB and ULAB locations,                    24APR89CHP
 1 \ compiles label data, reports errors
 2
 3 : COMPILE.ADDRESS  ( A---)
 4     LONG.ADDR @
 5     IF SP@ S0 = IF 0 THEN
 6        , ELSE C, THEN ;
 7
 8 : END PLACE.TAB BEGIN DUP @ WHILE DUP LAB.TAB FIND.LAB
 9  OVER OVER U. U. CR OVER OVER OR 0= IF ." Label " DROP DROP
10 DUP 5 TYPE ."  at " LAB.LEN + @ U. ." not found" CLEAR.TABLES
11 SP! QUIT THEN 1+ DUP C@ IF OVER OVER - 1- DUP ABS 7F > IF DROP
12 ."  Branch from " SWAP >R >R 5 TYPE ."  at " R> U. ."  to " R>
13   U. ." is too far" CLEAR.TABLES SP! QUIT THEN SWAP C! DROP
14 ELSE ! THEN LAB.LEN + 2+ REPEAT DROP CLEAR.TABLES ;
15  .( #5)   -->


ASSEMBLER SCR # 6
 0 \ CREATE operators for defining mnemonics                 24APR89CHP
 1 \ Multimode opcodes
 2 : M/CPU CREATE 2 ALLOT C, , DOES> 0 LONG.ADDR !     ?LEGAL
 3    ?ZP ?IMM
 4      C@ MODE C@ ADD.TABLE + C@ + C,  \ Adjust opcode
 5    MODE.KEY @ 20 = 0= IF
 6    COMPILE.ADDRESS THEN ZP ;
 7 \ Single-mode opcodes
 8 : CPU CREATE 2 ALLOT C, DOES>     C@ C, ZP ;
 9
10 : BRANCHES CREATE 2 ALLOT C, DOES>  C@ C, FF C, ZP ;
11
12
13
14
15 .( #6)   -->


ASSEMBLER SCR # 7
 0 \  Definitions of mnemonics                               24APR89CHP
 1 0060 61 M/CPU ADC, 0060 21 M/CPU AND, 0060 C1 M/CPU CMP,
 2 0060 41 M/CPU EOR, 0060 01 M/CPU ORA, 0060 E1 M/CPU SBC,
 3 0060 81 M/CPU STA, 0060 A1 M/CPU LDA,
 4 025E 02 M/CPU ASL, 025E 42 M/CPU LSR,
 5 025E 22 M/CPU ROL, 025E 62 M/CPU ROR,
 6 027E C2 M/CPU DEC, 027E E2 M/CPU INC,
 7 016F E0 M/CPU CPX, 016F C0 M/CPU CPY, 003F 14 M/CPU JSR,
 8 036D A2 M/CPU LDX, 016E A0 M/CPU LDY, 027D 82 M/CPU STX,
 9 007E 80 M/CPU STY, 007F 20 M/CPU BIT, 003F 40 M/CPU JMP,
10 00 CPU BRK, 18 CPU CLC, D8 CPU CLD, 58 CPU CLI, B8 CPU CLV,
11 CA CPU DEX, 88 CPU DEY, E8 CPU INX, C8 CPU INY, EA CPU NOP,
12 48 CPU PHA, 08 CPU PHP, 68 CPU PLA, 28 CPU PLP, 40 CPU RTI,
13 60 CPU RTS, 38 CPU SEC, F8 CPU SED, 78 CPU SEI, AA CPU TAX,
14 A8 CPU TAY, BA CPU TSX, 8A CPU TXA, 9A CPU TXS, 98 CPU TYA,
15 .( #7)  -->
```

```
STA FF00
JSR 0000
INX
INX
JMP NEXT
```

The dummy address FF00 guarantees that the absolute-address version of STA will be used, and the 00 is in the position that END inspects to decide on two-byte storage.

The FF00 addresses following STA must be changed to the two addresses *immediately following* the JSR—but JSR, automatically compiles two 00 bytes, hence -2 ALLOT is used to back up to the address for CALL1, then 1 ALLOT for the address of CALL2, then 1 ALLOT before compiling whatever follows.

An alternative technique is:

```
ASSEMBLE CALL 0 ,X LDA, ULAB
CALL1 FF00 STA, 1 ,X LDA, ULAB
CALL2 FF00 STA, 0 C, DLAB CALL1
1 C, DLAB CALL2 -2 ALLOT JSR,
INX, INX, ' NEXT 2+ JMP,
```

If a primitive needs to make reference to a Forth variable, the label can be associated with the creation of the variable. For example:

```
VARIABLE SOME
-2 ALLOT DLAB SOME
1 ALLOT DLAB SOME1
1 ALLOT
```

This provides for separate reference to the high and low bytes of the value of SOME, so that they can be addressed separately for modification purposes.

I have written this assembler for public-domain use with the Apple][family of computers. It occupies eight screens and will operate anywhere in memory.

Chester H. Page earned his doctorate in mathematical physics at Yale and spent some 36 years at the National Bureau of Standards. His first Forth was Washington Apple Pi's fig-FORTH, which he modified to use Apple DOS, then ProDOS, and later to meet the Forth-79 and Forth-83 Standards; more recently, he incorporated many features of F83.

```
ASSEMBLER SCR # 8
 0 \ More mnemonics and special definitions                  24APR89CHP
 1 90 BRANCHES BCC, B0 BRANCHES BCS, F0 BRANCHES BEQ,
 2 30 BRANCHES BMI, D0 BRANCHES BNE, 10 BRANCHES BPL,
 3 50 BRANCHES BVC, 70 BRANCHES BVS,
 4
 5 : GONEXT ['] NEXT 2+ JMP, ;
 6 FORTH DEFINITIONS
 7
 8 : DROP.ASSEMBLER 1 BEGIN 1+ DUP VOC.LIST @ NAME> ['] ASSEMBLER
 9    = UNTIL BEGIN DUP 1+ VOC.LIST @ DUP WHILE OVER VOC.LIST !
10    1+ REPEAT SWAP VOC.LIST ! ;
11 \ NOTE: DROP.ASSEMBLER not included in minimal HOST assembler
12 \ because VOC.LIST not in MIN.FORTH
13 : ASSEMBLE ?EXEC CREATE ASSEMBLER PRIM [ ASSEMBLER ] ZP !CSP ;
14
15 IMMEDIATE   .( #8)


ASSEMBLER SCR # 9
 0 \ Demonstration samples                                   10NOV89CHP
 1 HEX FORTH DEFINITIONS ASSEMBLER   CLEAR.TABLES
 2 ASSEMBLE (TEST) 0 # LDA, 80 # LDY, DLAB ONE 300 ,Y STA, DEY,
 3          ULAB ONE BPL, RTS,
 4 ASSEMBLE TRY 5 STX, 1 # LDX, DLAB TWO INX, 7 # CPX, ULAB TWO
 5       BNE, 5 LDX, ' NEXT 2+ JMP,
 6 ASSEMBLE HOME FC58 JSR, ' NEXT 2+ JMP,
 7
 8 \ Follow with a mispelled-label error
 9
10 ASSEMBLE THIS 0 # LDX, DLAB THREE 103 ,X LDA, ULAB FOUR BNE,
11   3 ,X STA,  ULAB ONE JSR, ULAB FIVE JMP, DLAB FOUR 104 ,X
12  STA,  DLAB FIVE INX, ULAB RHREE BNE, ' NEXT 2+ JMP,
13
14 \ Misspelled label: RHREE
15 END


ASSEMBLER SCR # 10
 0 \ Demonstration samples                                   10NOV89CHP
 1 HEX FORTH DEFINITIONS ASSEMBLER   CLEAR.TABLES
 2 ASSEMBLE (TEST) 0 # LDA, 80 # LDY, DLAB ONE 300 ,Y STA, DEY,
 3          ULAB ONE BPL, RTS,
 4 ASSEMBLE TRY 5 STX, 1 # LDX, DLAB TWO INX, 7 # CPX, ULAB TWO
 5       BNE, 5 LDX, ' NEXT 2+ JMP,
 6 ASSEMBLE HOME FC58 JSR, ' NEXT 2+ JMP,
 7
 8 \ Follow with a too-long branch error
 9
10 ASSEMBLE THAT DLAB THREE ULAB FOUR JMP, 2 ,X LDA, DLAB FOUR
11    0 # LDA, 80 ALLOT ULAB THREE BEQ,
12
13
14
15 END


ASSEMBLER SCR # 11
 0 \ Demonstration samples - duplicate label                 10NOV89CHP
 1 HEX FORTH DEFINITIONS ASSEMBLER   CLEAR.TABLES
 2 ASSEMBLE (TEST) 0 # LDA, 80 # LDY, DLAB ONE 300 ,Y STA, DEY,
 3          ULAB ONE BPL, RTS,
 4 ASSEMBLE TRY 5 STX, 1 # LDX, DLAB TWO INX, 7 # CPX, ULAB TWO
 5       BNE, 5 LDX, ' NEXT 2+ JMP,
 6 ASSEMBLE HOME FC58 JSR, ' NEXT 2+ JMP,
 7
 8 \ Follow with a duplicate-label error
 9
10 ASSEMBLE THIS 0 # LDX, DLAB THREE 103 ,X LDA, ULAB TWO BNE,
11   3 ,X STA,  ULAB ONE JSR, ULAB FIVE JMP, DLAB TWO 104 ,X
12  STA,  DLAB FIVE INX, ULAB THREE BNE, ' NEXT 2+ JMP,
13
14 END
15
```

(Figure One on page 41.)

Part Three

FORST: A 68000 NATIVE-CODE FORTH

JOHN REDMOND - SYDNEY, AUSTRALIA

In this installment of my series on ForST for the Atari ST, I want to describe an implementation of a much-discussed approach to the use of local (automatic) variables [Ros87, Yli89, Bet89, Hay89]. The nice outcome of the ForST implementation is that it has little in the way of space and speed penalties.

I illustrate the approach with an implementation of a package for arithmetic using IEEE short reals and, for a little light relief after that, the Sieve and Towers of Hanoi. The short reals have the advantage of being the same size as the ForST 32-bit integers, so all stack operations—fetches and stores, as well as the numeric comparisons—are the same.

Code with more than two SWAPs or one ROT is faster with locals.

The compiled code created by ForST very much follows the practice of C by using the LINK and UNLK instructions of the 68000. LINK pushes the contents of a specified register (A4 in our case) onto the hardware stack, saves the value of the stack pointer (A7) in that register (A4), and then increases the value in A7 by an immediate value (usually negative). The result is allocation of an area for local variables on the return stack which can be accessed by offset addressing using register A4. This area is totally secure and the code is therefore fully reentrant. UNLK deallocates the area and restores the value in A4.

The special words associated with the local variables are:

```
( ******* Floating-point arithmetic words *******)

Copyright © 1989, John Redmond,
23 Mirool St, West Ryde, NSW 2114, Australia.

Written permission is required for commercial use.

hex
: q/  { 2 args num den }
  0 num den 0  ud/mod  2swap 2drop ( 64-bit quotient) ;

: align   { 1 arg exponent }
  ( double mantissa left on the stack)
  2dup or
  if  ( non-zero)
    begin
      dup ff800000 and  0=
    while
      d2*  -1 addto exponent
    repeat
    swap 0< -  dup 1000000 < not
    if
      2/  1 addto exponent
    then  exponent
  then  ( mantissa,exponent) ;

: expon  17 lsr  0ff and 7f - ( fno exponent) ;
: mant  7fffff and 800000 or ( absolute mantissa) ;
: smant  dup mant  swap 0< if negate then ( signed mantissa) ;

: pack  { 2 args mantiss exp }
  mantiss ( test if non-zero)
  if
    mantiss abs 7fffff and
    exp 7f + 0ff and wflip 7 lsl  or
  else 0
  then ( real number) ;

: fnegate  dup 0= not  if  80000000 xor then ;
: fabs  7fffffff and ;

: f*  { 2 args fn1 fn2 }
  fn1 0= fn2 0= or
  if 0  ( zero product)
  else
```

{

Sets state to zero, compiles the code for LINK A4,#0, saves the address of the zero field for later patching, and sets the total size of the local area to zero.

}

Sets state to -1, ready for compilation.

LOCAL and LOCALS (synonyms)
Expect the number of local variables on the stack and the names of the local variables directly following in the input stream, so that the local headers can be set up in normal header space. As each local variable is added, the size variable is decremented by four.

ARG and ARGS
Perform the same functions as LOCALS and, in addition, compile code to move the arguments from the data stack into local space on the return stack, using the code:

```
LEA   -(#ARGS*4)(A4),A0
```

followed by a series of:

```
MOVE.L   (A6)+,(A0)+
```

instructions (one for each argument).

LOCBUFF
Expects a buffer size on the data stack and adds this value to the local buffer. 32 LOCBUFF MYPAD reserves MYPAD as a named 32-byte buffer in stack space.

The accompanying code examples illustrate the use of the local words. The most important requirements are that they be used between { and }, and that ARG or ARGS be used before LOCAL, LOCALS, or LOCBUFF.

The following words direct the code to access the stack variables:

TO
Sets a compilation flag to compile a move from the data stack TO the local area (the default mode is FROM the area to the data stack). ADDTO sets the flag to compile code to add a value on the top of the data stack to the data area. ADDR sets the flag to compile code to push the address of a local variable to the data stack.

A philosophical disadvantage of these three words is that they muddy the postfix water, but Forth is not pure in this respect anyway (CREATE and COMPILE are examples of the built-in aberrations).

```
    fn1 mant fn2 mant um*  ( mantissa product)
    fn1 expon fn2 expon + 9 +  align  pack
    fn1 fn2 xor 0< ( net sign)
    if fnegate then
  then  ( real number) ;

: f/ { 2 args num den }
  den 0=
  if  7fffffff ( infinity)
      num 0< if fnegate then
      exit
  then

  num 0=
  if  0  exit then          ( zero)

  num mant den mant q/  ( mantissa-quotient)
  num expon den expon -  17 + align  pack
  num den xor 0<
  if fnegate then  ( real-quotient) ;

: +mants { 3 args #shifts bigger smaller }
  #shifts 17 >
  if
    bigger smant ( smaller fno just too small)
  else
    smaller smant #shifts asr
    bigger smant +
  then ( mantissa sum) ;

: f+  { 2 args fn1 fn2  3 locals diff exponent fsign }
  fn1 0= if  fn2  exit then ( don't waste time)
  fn2 0= if  fn1  exit then

  fn1 expon  fn2 expon  -  to diff
  fn1 expon to exponent ( larger exponent?)

  diff
  if
    diff 0>
    if
      diff fn1 fn2
    else
      fn2 expon to exponent
      diff abs fn2 fn1
    then
    +mants
  else
    fn1 smant fn2 smant + ( add if expons equal)
  then

  dup 0= if exit then
  dup to fsign  abs ( mantissa)

  dup 800000 <
  if
    begin
      2*  -1 addto exponent
      dup 800000 and
    until ( left justified)
  else
    begin
      dup ff000000 and
    while ( still overflowing on left)
      2/  1 addto exponent
```

; and EXIT
These are enhanced to compile an UNLK A4 instruction before the usual RTS. ; also carries out the patch of the link field to allocate the correct total space, and sheds the headers for the local variables.

A consequence of the use of local variables is an almost automatic change in programming style. The definition of F+ (which illustrates my current explorations into extended Forth style) is a good example, in that it rivals some of the very large definitions in the C or Pascal literature. Such code would not be possible in standard Forth (and is arguably not a good idea in any language!). Particular attention is drawn to the use of pointers in FCONVERT, which allow it to alter stack variables of the higher word, FNUMBER. I find the syntax much less confusing than that of C and Pascal.

Despite the size of the definitions, they are reasonably easy to understand and maintain—and it suggests a use of extended Forth as a postfix algorithmic language, which is the ultimate test of code clarity.

The floating-point package works well, particularly if it is compiled in the MACRO mode. The 32-bit values consist of the following fields:

bits 0–22: unsigned mantissa (bit 23 assumed to be 1)
bits 23–30: exponent biased by +127
bit 31: mantissa sign

The precision is, of course, limited by the 24-bit mantissa, but gives quite good seven-figure decimal accuracy. Extended calculations will lead to some loss of accuracy through repeated rounding. This can be avoided by using a special stack for extended temporary reals. The coding approach is sufficiently generic for it to be extended to doubles, but not many of us really need them. Short reals have enough range and precision for Avogadro's number and Planck's constant, and that's good enough for me!

I do not propose to get involved in an explanation of how the package works. That is outside the scope of these articles. I can say, though, that writing it gave me a very strong feeling for the inherent imprecision of reals—and for the considerable overhead of working with them. This is

```
    repeat
  then ( exponent)

  exponent pack
  fsign 0<
  if fnegate then ( signed real sum) ;

: f-  fnegate f+ ;

: i>f  { 1 arg numb }
  numb 0=
  if 0
  else
    numb abs 0  37 align  pack
    numb 0<
    if fnegate then
  then ;

: f>i { 1 arg fno  1 local exponent }
  fno 0=
  if 0
  else
    fno expon to exponent
    exponent 0<
    if
      0
    else
      8 addto exponent
      fno mant 0 ( make double)
      begin
        d2*  -1 addto exponent
        exponent 0<
      until
      swap drop
      fno 0<  if negate then
    then
  then ;

1 i>f constant f1.0
0a i>f constant f10.0
f1.0 f10.0 f/ constant f0.1
f1.0 2 i>f f/ constant f0.5

: fix  f>i i>f ;
: int  dup fabs  f0.5 f+ fix
  swap 0< if fnegate then ;
: fmod  2dup f/ fix f* f- ;

( ******* Floating-point output words *******)

decimal

: normalize { 1 arg fno  1 local decexp }
  -1 to decexp
  fno expon 0<
    if  fno fabs
      begin dup f0.1 <
      while f10.0 f*  -1 addto decexp repeat
    else  fno fabs
      begin dup f1.0 < not
      while f0.1 f*  1 addto decexp repeat
    then
  decexp ( normalized mantissa, dec exponent) ;
```

```
: spill
  dup mant  swap expon 8 + lsl
  40000000 um*  swap
  0< if 1+ then 2/ ( decimal mantissa) ;

: sigfigs  { 1 arg mantissa  2 locals #digs quotient }
  mantissa  7 to #digs
  begin
   dup to quotient
   10 /mod swap  0=  #digs 1 > and
  while
   -1 addto #digs
  repeat  drop
  quotient #digs ( scaled mantissa, #digits) ;

: decpt  46 hold ;
: figures  dup 0> if  0 do # loop  else drop then ;
: zeros  dup 0> if  0 do 48 hold loop  else drop then ;

: scientific  { 4 args fno value length exponent }
  exponent abs
  <#
    #s  exponent sign  69 hold  drop
    value
    length 1 >
    if
      length 1- figures decpt
    then
    #  fno sign
  #>  ( string,length) ;

: vsmall  { 4 args fno value length exponent }
  value
  <#
    #s
    exponent -1 <
    if exponent abs 1- zeros then
    decpt 1 zeros  fno sign
  #>  ( string,length) ;

: small  { 4 args fno value length exponent  1 local diff }
  exponent length - to diff
  value
  diff 0=
  if
   <# 1 zeros  #s  fno sign  #> exit
  then

  diff 0>
  if
    <# diff 1+ zeros  #s  fno sign #>
  else
    diff -1 =
    if
      <# #s fno sign #>
    else
      <# diff abs 1- figures decpt  #s  fno sign #>
    then
  then ( string,length) ;

: (f.) { 1 arg fno  3 locals decexp numb numbase }

  base @ to numbase  decimal ( force to decimal output)
```

(Continued on page 32.)

nowhere more evident than in the code for the output word `(F.)`. It is very complicated and, often, the result is only approximate.

The floating-point arithmetic is about ten times slower than equivalent integer code (which surprises no one), but it can be made up to three times faster by reworking in assembly code. The transcendental definitions have not been included because of space constraints, but they are easily developed using series expansions [Koo87] and Horner's rule.

Time and Space

To close, some comparisons between code with and without local variables. The DOPRIME word of the sieve benchmark takes 202 bytes (196 bytes in the normal form) and the execution time is the *same*. In other words, the time for fetches and stores from or to the local stack space ends up essentially the same as that for OVERs, DUPs, and DROPs. I am *not* recommending the use of locals as a panacea for all definitions, but any code which includes more than two SWAPs or a single ROT will work faster with them. I am sold on the approach and intend to use it in all future large projects.

References

[Ros87] P. Ross, "Local Variables," *Forth Dimensions* (IX/4).

[Bet89] J. Betancourt, "Prefix Frame Operators," Forth Dimensions (XI/1).

[Hay89] J.R. Hayes, "Local Variables, Another Technique," *Forth Dimensions* (XI/1).

[Koo87] P. Koopman, Jr., "Transcendental Functions," *Forth Dimensions* (IX/4).

[Yli89] J. Yli-Nokari, "Local Variables and Arguments," *Forth Dimensions* (XI/1).

John Redmond is an Associate Professor of Organic Chemistry at Sydney's Macquarie University. He is a "...sometimes-evenings-when-I-have-time programmer" whose chief disappointment of 1988 consisted of attending a plant pathology conference in Acapulco while Forth's own Charles Moore was visiting Sydney. Mr. Redmond welcomes letters from FD readers: 23 Mirool Street, West Ryde, NSW 2114, Australia.

BEST OF GENIE

GARY SMITH - LITTLE ROCK, ARKANSAS

News from the GEnie Forth RoundTable—Some glaring truths emerge even under the most casual observation. One such truth is there is a wealth of opportunity to learn C via formal education, but little such opportunity presents itself for one wishing to learn Forth.

As two guest conferences dedicated to "Teaching Forth" indicate (Mahlon Kelly, July 1988; John Wavrik, March 1990), the landscape is not totally barren. Several vendors and other guests, such as Gary Feierbach, have also noted their teaching efforts. But since this discussion is with regard to academic instruction, we will have to save those insights for another visit.

One of the more active educators involved in the teaching of Forth is Jack Brown, of the British Columbia Institute of Technology and sysop of the BCFB link in ForthNet. It is Jack's excellent F-PC tutorial that is carried in Category 15 on GEnie.

So you do not labor under the false impression that Jack is the *only* Forth educator, let's look in Category 2, Topic 2 as Jack, Mark Smiley, and Archie Warnock discuss Mark's upcoming Forth classes which will use F-PC for the course model.

I have also prefaced this discussion with a course announcement that may be of some interest, since it is centered on a single-chip engine, the 68HC11. It is too late to get involved in this course, as announced, but I am satisfied that letters to John Schoonover will garner a courteous response regarding future course offerings.

Topic 2
Forth Taught at Schools/Universities
News, methods, and collaboration ideas for those teaching Forth at schools, colleges, and universities.

To: All
From: Jack Brown
Subj: 68HC11-based course

Those within commuting distance of the British Columbia Institute of Technology may be interested in the following course offered by the Instrumentation and Control Option of the Electrical and Electronics Program at BCIT.

Course:	Introduction to Single-Chip Microcontrollers (68HC11 used in course)
Dates:	Thursdays, starting February 8, 6:45 p.m. to 9:45 p.m. (10 weeks)
Cost:	$210 includes 68HC11 reference manual, pocket guide, data sheets, and lab handouts.

I was talking to the instructor, Mr. John Schoonover, today and here is some information about the course:

- Course is 60% lecture and 40% lab.
- All major I/O functions and devices of the 68HC11 will be discussed and exercised.
- Programming is in assembly language and begins with pulse generation and pulse with modulation, which I am told can be used to turn an output port bit into an analog output (D/A).
- Final project consists of turning the 68HC11 into an eight-channel analog scanner (using the HC11's A/D ports). Test data for the scanner is generated by using the 68HC11 as an eight-channel D/A output generator.

Please note that the above comments are my recollections from talking to John and are not meant to be a complete course description. It looks like there might be at least three members of BC FIG attending this course: John Somerville, Jack Brown, and Dave Brown. John Schoonover has said that it should be possible for us to do all of the lab exercises in Forth (if we choose to do so), although assembly language is currently used in the course.

You can register for this course by calling BCIT's extension division (604-434-1610). The course registration number (CRN) is 08950 and the title is "Introduction to Single-Chip Micro Controllers."

To: Jack Brown
From: Mark Smiley
Subj: Teaching F-PC

I am going to be teaching a seven-week course in Forth next Fall at Goucher College using F-PC. The students will all have completed a seven-week course in Fortran. Is there a book you would recommend for the class? I need to choose a book soon.

I have taught Forth twice before. Once, I used MVP-FORTH with *Starting Forth* (1st ed.), then I used F83 with *Mastering Forth*. I was thinking of using either *Mastering Forth*, or *Starting Forth* (2nd ed.), but I'm open to other ideas (like Kelly & Spies).

I understand you teach Forth using F-PC. What book do you use? Did you require students to also purchase a copy of Ting's *System Guide to F-PC*? How did you work the cost of the program (did you get a site license from Tom Zimmer, or buy enough copies for the machines at your school, or what)?

Thanks for the advice.

To: Mark Smiley
From: Jack Brown
Subj: Teaching F-PC

I am using a revised set of notes for F-PC 3.5, which were upgraded from the tutorial on the BBS. Actually the revision process is not yet complete, as I am converting them as this course progresses. These notes are in the public domain and can be revised or modified further by anyone wishing to use them. I have not assigned a required text but suggest the students get a copy of *Starting Forth* 2nd ed. or *Mastering Forth.*

F-PC 3.5, according to the notice, is public domain and not shareware, except for those pieces where contributors have restrictions. So I just gave each student a copy.

In my course I am not going too deeply into the internals of F-PC. Instead, we will spend the last week studying the source for Zen 1.1, which is much simpler to understand! The version I will be using was sent to me by Wil Baden and can be downloaded from BCFB as WB1ZEN11.ZIP and WB2ZEN11.ZIP. It is a fairly complete system and includes many of Wil's extras.

P.S. I am enjoying the graphics package!

To: Mark Smiley
From: Jack Brown
Subj: Teaching F-PC

"I have recently acquired copies of LESSON0.ZIP - LESSON5.ZIP from ECFB. I assume this is the tutorial you mentioned. Have I got it all?"

There are ten lessons in all... I only uploaded the first five. I am currently upgrading lesson six to F-PC 3.5 and should have finished all ten by the middle of May. I could send you (mail) the first five or six on disk if you would like to take a look. My students are also working out solutions to all the problems. Some of the lessons have up to 30 problems that require solutions. Most of these are fairly simple and straightforward. From my experience, most of my students seem to learn Forth best by having lots of examples to try with problems that suggest modifications to be made or variations to be implemented. (That is, unless they are brilliant self-starters!)

"When you have finished the current revision, are you planning to put it on the BCFB/ECFB?"

I talked to Jerry Shifrin the other day

(Continued from page 22.)

`CREATE WORKAREA 82 ALLOT` (Kernel)
Create a work area to hold 80 characters and two extra bytes.

`EXPECT ( a l -- )` (Kernel)
Get *l* characters from the input device. Input halts when the full number have been received, or when CR is pressed. A certain amount of editing is allowed. When done, a variable named `SPAN` contains the number of bytes actually received.

`GETINPUT ( a l -- )` (Screen 2)
Use `EXPECT` to get up to *l* characters from the input device, and store them at the address *a* as a counted string. (In a *counted string*, also called a *memory* or *packed string*, the first byte holds its length, and the remaining bytes its data.) Then append a blank, which is not included in the count. The blank is necessary for number conversion to work properly.

`-LEADING ( a l -- a' l' )` (Screen 2)
Trim leading blanks, analogous to the kernel word `-TRAILING` that trims trailing blanks.

`: LEX` (Screen 2)
`( a l b -- taila taill heada headl )`
Scan forward into a string until a separator byte *b* is found. That byte and the remainder of the string become the "tail" and the first part becomes the "head." If the first byte is a separator, then the head is null (i. e., zero length), and the same is true if the end of the string has been reached. (Some versions of `LEX` reverse the order of the head and tail in the stack diagram.)

`NUMBER? ( a -- d f)` (Kernel)
This word, not available in all Forths, is used internally by F83 to convert a string to a number. It is rather specialized, as it was created to meet the needs of `INTERPRET`, and was not designed for everyday use. Still, we are free to employ it—Forth does not distinguish between a kernel word and a user-created one.

`NUMBER?` takes a counted string at the address *a*, and begins to convert it to a number using the current base, which is contained in the variable `BASE`. The base may be any number, not limited to 2, 8, 10, and 16. The address *a* points to the count byte, but this is ignored and conversion starts on the next byte. Any non-numerical byte will stop conversion, with two exceptions: A leading minus sign makes the number negative. A dot sets an internal variable named `DPL` (Decimal PLaces), but otherwise has no effect. On exit, a double-precision (32-bit) number and a flag are left on the stack. The flag is true if conversion was terminated by a blank, and is false otherwise.

`ADD'EM ( -- )` (Screen 2)
Get a line, extract numbers, and display their sum. Continue until the user enters a blank line. Note the phrase `BL LEX DROP 1-` which adjusts the stack to the requirements of `NUMBER?`.

Figure One. Detailed documentation for Screen Two.

(Redmond code, continued from page 29.)

```
  fno 0=
  if
    0  <# # #>
  else
    fno normalize ( mant,decexp) to decexp
    ( mantissa) spill to numb
    fno ( sign)  numb sigfigs ( mantissa, #digits)  decexp

    decexp -3 < decexp 5 > or
    if
      scientific
    else
      decexp 0<
      if vsmall else small then
    then

  then ( string,length)

  numbase base ! ;

: f. (f.)  type space ;

( ******* Floating-point input words *******)

decimal

: fconvert
 { 3 args &pointer &value &range  2 locals pointer char }

  &pointer @ 1+ to pointer

  begin
    pointer c@ to char
    char 47 > char 58 < and
  while
    &value @ f10.0 f*  char 48 - i>f f+ &value !
    &range @ 0< not
    if 1 &range +! then
    1 addto pointer
  repeat

  pointer  &pointer ! ;

: expconvert  { 1 arg pointer  3 locals char numb sign }
  1 addto pointer  0 to numb  0 to sign

  pointer c@ dup 45 = ( minus sign)
  if  drop 1 addto pointer  -1 to sign
  else
    43 = ( plus sign)
    if  1 addto pointer  then
  then

  begin
   pointer c@ to char
   char 47 > char 58 < and ( digit)
  while
   numb 10 *  char 48 - +  to numb
   1 addto pointer
  repeat ( absolute integer exponent)

  numb  sign if negate then ( signed integ expon) ;

: fnumber  { 1 arg charptr  3 locals sign mantissa dpl }
```

and we were tossing around the idea of posting the revised tutorial lessons for F-PC 3.5 in the MetroNet Forth Conference to see what kind of response it would get from a wider distribution. Who would like to see this happen?

> "I am unaware of anyone's restrictions, except on Bob Smith's SFLOAT.SEQ, and on your (& the Kent Bros.) VPSFLOAT.SEQ. But as I understand it, neither of these packages would not allow us to give copies to students. So I assume you gave the students the complete package, eh?"

I use the VP-Planner floating point exclusively. Bob's package is restricted to personal use and I assume that's what the students will be doing with it, although I will not be using it in the course. My students are using a computer lab with IBM Model 70s installed (386s with VGA and your graphics package works fine on these). They have a software protection system installed by the school, called Integrity, which prevents students from accessing the installed software. I had our computer systems department install F-PC 3.5 on all the machines in the lab under a password, so only Forth students can use Forth—as it would be possible for a clever (or not so bright) Forth programmer using F-PC to violate Integrity's security.

All of my Forth students have some type of PC with hard disk at home, on which I presume they will install F-PC.

> "It seemed fairly useless, since there was almost no documentation, and you couldn't save the system or use assembler (except crudely, via C,). I take it that it's a usable system now? Is there any documentation these days?"

There is still no *.DOC file, but there are many extensions with good documentation in the *.SCR files. I have not used it extensively and am mainly interested in it because the 1.1 version is Forth-83 and the Kernel Source is mostly high level and has been kept quite simple. An ideal system if you want to study the structure of the Forth compiler/interpreter as part of your course.

> "I used to use his old F83X, before I switched to F-PC. I hear he has a new version of F83X out now (3.0). Have you tried that?"

I have not used Wil's F83X. Is this 3.0 version recent? It is odd that Wil didn't mention it to me when I was talking to him a few weeks ago. Perhaps someone else upgraded it. To be quite honest, I have one H*[[of a time keeping up with all the Forths available, and must confess that there are many I have not tried!

One other comment on my Forth students and my choice of F-PC.

Earlier (last Fall), I posted a note that I would be teaching this Forth course and asked people which Forth I should use. From the replies, the consensus seemed to be that a small system like Zen, Guy Kelly's FORTH83, or Pygmy might be the best way to go. I choose F-PC because I wanted a system that used text files and one that would make a "good impression" on computer science students that have used dozens of applications packages and may already know half a dozen languages!

F-PC and its environment were a good choice... My students had just completed a SmallTalk course on the PS/2 386 Model 70s, and it ran like a slug! (Maybe an exaggeration on my part, but they were impressed with the speed and interactivity of F-PC and the hypertext!) I think it would have been embarrassing to try and get them to use a small Forth system with a block editor.

Dr. Richard Haskell of Oakland University in Rochester, New York, is also teaching an introductory course using F-PC 3.5. He is also working on a set of notes that will be available on disk. He said he is bringing his notes, hopefully for distribution, to the Rochester Forth Conference this June.

I will be also attending Rochester Conference this June. Why don't you join us?

To: Jack Brown
From: Mark Smiley
Subj: Teaching F-PC

"I am currently upgrading Lesson 6 to F-PC 3.5 and should be finished with all ten by the middle of May..."

I'm willing to wait till all ten are done. But I'd love to get a copy of the full set then.

"From my experience most of my students seem to learn Forth best by having lots of examples to try with problems..."

(Redmond code, continued.)

```
  0 to sign  0 to result  -1 to dpl

  charptr 1+ c@ 45 = dup
  if  drop 1 addto charptr  -1 to sign
  else
    43 =
    if  1 addto charptr  then
  then

  addr charptr  addr result  addr dpl  fconvert
  charptr c@ 32 =
  if mantissa exit then ( just an integer input)

  charptr c@ 46 = ( decimal point)
  if
    0 to dpl
    addr charptr  addr result  addr dpl  fconvert
  then

  dpl 0< if 0 to dpl then ( no scaling needed)

  charptr c@ 95 and 69 = ( E)
  if
    charptr expconvert negate  addto dpl
  then

  result ( ready for scaling)
  dpl 0>
  if
    begin  dpl 0>
    while  f0.1 f*  -1 addto dpl
    repeat
  else
    begin  dpl 0<
    while  f10.0 f*  1 addto dpl
    repeat
  then ( scaled result)

  sign  if  fnegate  then ;

( fetch fnumber from input stream)
: ffetch  32 word fnumber ; immediate

(Sieve benchmark:)

decimal

8190 constant size
create flags size allot

: do-prime

{  3 locals p q cnt }

   flags size 1 fill

   0 to cnt
   size 0
   do
      flags i + c@
      if i dup + 3 +  to p
         p i +  to q
         begin  q size <
         while
```

(Continued on next page.)

(Redmond code, continued.)

```
            0 flags q + c!
            p addto q
         repeat
         1 addto cnt
      then
   loop

   cnt . ." Primes " ;

: primes  0 do do-prime loop ;

( example:  25 primes)

(Towers of Hanoi:)

: movedisc
{ 2 args frompeg topeg }
  frompeg . ." to " topeg .  2 spaces ;

: movetower
{ 4 args height frompeg topeg usingpeg }
    height if
         height 1- frompeg usingpeg topeg recurse
         frompeg topeg movedisc
         height 1- usingpeg topeg frompeg recurse
    then ;

: hanoi { 1 arg  height }
    cls  height 1 3 2 movetower ;

( example: 6 hanoi )
```

That was part of my experience as well, though I also had my students do a major project at the end of the course.

"...the idea of posting the revised tutorial lessons in the MetroNet Forth Conference... Who would like to see this happen?"

Me, though you should also put a message about it in the F-PC conference when you do, since some of us don't subscribe to the Forth conference.

"I have not used Wil's F83X... Perhaps someone else upgraded it."

Could be. I noticed a fairly recent upload on ECFB, but I haven't downloaded it yet.

"I have one H*[[of a time keeping up with all the Forths..."

Me too!

"...it would have been embarrassing to try and get them to use a small Forth system..."

I agree totally.

"Why don't you join us?"

I would like to, but I doubt I can this year. I am currently just a poor graduate student. I used to teach full-time at AUM (where I last taught Forth), then I went back to school to get my Ph.D. in math, which I hope to acquire this June. Next year, though, I may well be able to get Goucher to send me to the conference. I have been to FIG and FORML meetings, but never to Rochester.

"Dr. Richard Haskell is also working on set of notes..."

I would be very interested in copies of both his and your notes, when complete. By the way, have you ever used the graphics routines in your course? If so, I'd be interested in the problems you designed for it. I've been toying with the idea that, since Forth is ideal for process control, I might use the turtle graphics as a sort of robot that could be programmed for simple tasks. It would also give them something interesting to look at.

To: Mark Smiley
From: Steve Palincsar
Subj: Teaching F-PC

Well, considering the fact that all your students will have completed a course in Fortran, I should think you'd give *very* serious consideration to Kelly & Spies, since the others you mention begin at a very much lower level—and also contain much less information.

To: Mark Smiley
From: Jack Brown
Subj: Teaching F-PC

"Have you ever used the graphics routines in your course?"

I used it in lesson four to plot random dots for the visual testing of various random-number generators. I gave the students the whole package, and they were also running your demos.

To: Mark Smiley
From: Archie Warnock
Subj: Teaching F-PC

I found that my students were very receptive to the idea of learning to write a program which would crash the machine. (Seriously!) It gave them an appreciation for the power of the language and also helped overcome the initial intimidation at the prospect of a new and completely different type of language. They had a great time.

And for image processing—a good excuse for looking at pretty pictures!

To: Archie Warnock
From: Mark Smiley
Subj: Teaching F-PC

"...mine were very receptive to the idea of learning to write a program which would crash the machine."

Which of the many ways did you suggest? Or did you let them try to figure it out themselves? How much Forth had you taught them at that point? Did they know how to do loops in Forth?

(Continued on page 39.)

Testing Toolkit

Corrections to code printed in FD *XII/3*

Our last issue featured Phil Koopman Jr.'s interesting "Testing Toolkit." Apparently, one of our file filters ran amuck and introduced mistakes in the code. Our apologies to the author and to any readers who were baffled by our error.

Rather than describe the permutations of single, double, and triple hyphens that were affected, we decided to reprint the correct code here in its entirety. Also, the last two lines of the code example in the text of Koopman's article should have been:

```
--- )RS
--- 1111 1111 )DS
```

```
\ Forth testing support
\ By Philip Koopman Jr., for Harris Semiconductor
\   Derived from test code used for the RTX chip family
\   Developed on F-TZ (an F-PC and F-83 derivative) version 3.X11

VARIABLE #STACK  -1 #STACK !     \ Saves number of stack elements for testing
CREATE R-SAVE 8 ALLOT            \ Note: F-TZ uses 32-bit return addresses!

: GET-DEPTH  ( ..stack.stuff.. - ..stack.stuff.. )
  DEPTH  #STACK @ -  #STACK ! ;

: DS(    ( - $BAD1 $BAD2 )
  \ Init RS to -1 so that '--' will know it is a DS input
  \ Uses hex 0BAD1 and hex 0BAD2 as sentinel values for DS
  -1 #STACK !     $BAD1 $BAD2 ;

: RS(    ( - $BAD3 $BAD4 )
  \ Uses hex 0BAD3 and hex 0BAD4 as sentinel values for RS
  DEPTH #STACK !  $BAD3 $BAD4 ;

: --     ( n1 n2 n3 .. n.n - n1 n2 n3 .. n.n sentinel )
  #STACK @  0<  NOT    IF ( if RS( )  GET-DEPTH   THEN ;

: ?DATA    ( n1 n2 -- )
  = NOT  ABORT" DATA STACK ERROR" ;

: ?RETURN  ( n1 n2 -- )
  = NOT  ABORT" RETURN STACK ERROR" ;

: ---     ( -)
  DEPTH  #STACK !  ;

: PERCOLATE  ( r1 n.n .. n1 --  n.n .. n1 r1 )
  #STACK @  ROLL  -1 #STACK +! ;

: )RS        ( r.n .. r3 r2 r.1  n1 n2 n3 .. n.n - )
  GET-DEPTH   #STACK @
  IF  BEGIN  PERCOLATE  ?RETURN   #STACK @ 0= UNTIL  THEN
  $BAD4 ?RETURN  $BAD3 ?RETURN  -1 #STACK !  ;

: )DS        ( r.n .. r3 r2 r.1  n1 n2 n3 .. n.n - )
  GET-DEPTH   #STACK @
  IF  BEGIN  PERCOLATE  ?DATA     #STACK @ 0= UNTIL  THEN
  $BAD2 ?DATA    $BAD1 ?DATA    -1 #STACK !  ;

: REVERSE    ( n.1 n.2 .. n.n  n  --  n.n .. n.2 n.1 )
  DUP 0> IF  0 DO   I ROLL   LOOP  ELSE DROP THEN ;
```

(Continued on page 39.)

REFERENCE SECTION

Forth Interest Group

The Forth Interest Group serves both expert and novice members with its network of chapters, *Forth Dimensions*, and conferences that regularly attract participants from around the world. For membership information, or to reserve advertising space, contact the administrative offices:

Forth Interest Group
P.O. Box 8231
San Jose, California 95155
408-277-0668

Board of Directors

Robert Reiling, President *(ret. director)*
Dennis Ruffer, Vice-President
John D. Hall, Treasurer
Wil Baden
Jack Brown
Mike Elola
Robert L. Smith

Founding Directors

William Ragsdale
Kim Harris
Dave Boulton
Dave Kilbridge
John James

In Recognition

Recognition is offered annually to a person who has made an outstanding contribution in support of Forth and the Forth Interest Group. The individual is nominated and selected by previous recipients of the "FIGGY." Each receives an engraved award, and is named on a plaque in the administrative offices.

1979 William Ragsdale
1980 Kim Harris
1981 Dave Kilbridge
1982 Roy Martens
1983 John D. Hall
1984 Robert Reiling
1985 Thea Martin
1986 C.H. Ting
1987 Marlin Ouverson
1988 Dennis Ruffer
1989 Jan Shepherd

ANS Forth

The following members of the ANS X3J14 Forth Standard Committee are available to personally carry your proposals and concerns to the committee. Please feel free to call or write to them directly:

Gary Betts
Unisyn
301 Main, penthouse #2
Longmont, CO 80501
303-924-9193

Mike Nemeth
CSC
10025 Locust St.
Glenndale, MD 20769
301-286-8313

Andrew Kobziar
NCR Medical Systems Group
950 Danby Rd.
Ithaca, NY 14850
607-273-5310

Elizabeth D. Rather
FORTH, Inc.
111 N. Sepulveda Blvd., suite 300
Manhattan Beach, CA 90266
213-372-8493

Charles Keane
Performance Packages, Inc.
515 Fourth Avenue
Watervleit, NY 12189-3703
518-274-4774

George Shaw
Shaw Laboratories
P.O. Box 3471
Hayward, CA 94540-3471
415-276-5953

David C. Petty
Digitel
125 Cambridge Park Dr.
Cambridge, MA 02140-2311

Forth Instruction

Los Angeles—Introductory and intermediate three-day intensive courses in Forth programming are offered monthly by Laboratory Microsystems. These hands-on courses are designed for engineers and programmers who need to become proficient in Forth in the least amount of time. Telephone 213-306-7412.

On-Line Resources

To communicate with these systems, set your modem and communication software to 300/1200/2400 baud with eight bits, no parity, and one stop bit, unless noted otherwise. GEnie requires local echo.

GEnie

For information, call 800-638-9636

- Forth RoundTable
  (ForthNet link)*
  Call GEnie local node, then type M710 or FORTH
  SysOps: Dennis Ruffer (D.RUFFER), Scott Squires (S.W.SQUIRES), Leonard Morgenstern (NMORGENSTERN), Gary Smith (GARY-S)
- MACH2 RoundTable
  Type M450 or MACH2
  Palo Alto Shipping Company
  SysOp: Waymen Askey (D.MILEY)

BIX (ByteNet)

For information, call 800-227-2983

- Forth Conference
  Access BIX via TymeNet, then type j

forth
Type FORTH at the : prompt
SysOp: Phil Wasson (PWASSON)
- LMI Conference
Type LMI at the : prompt
Laboratory MicroSystems products
Host: Ray Duncan (RDUNCAN)

CompuServe
For information, call 800-848-8990
- Creative Solutions Conference
Type !Go FORTH
SysOps: Don Colburn, Zach Zachariah, Ward McFarland, Jon Bryan, Greg Guerin, John Baxter, John Jeppson
- Computer Language Magazine Conference
Type !Go CLM
SysOps: Jim Kyle, Jeff Brenton, Chip Rabinowitz, Regina Starr Ridley

Unix BBS's with forth.conf (ForthNet links and reachable via StarLink node 2533 on TymNet and PC-Pursuit node dcsfa on TeleNet.)*
- WELL Forth conference
Access WELL via CompuserveNet or 415-332-6106
Fairwitness: Jack Woehr (jax)
- Wetware Forth conference
415-753-5265
Fairwitness: Gary Smith (gars)

PC Board BBS's devoted to Forth (ForthNet links)*
- East Coast Forth Board
703-442-8695
StarLink node 2262 on TymNet
PC-Pursuit node dcwas on TeleNet
SysOp: Jerry Schifrin
- British Columbia Forth Board
604-434-5886
SysOp: Jack Brown
- Real-Time Control Forth Board
303-278-0364
StarLink node 2584 on TymNet
PC-Pursuit node coden on TeleNet
SysOp: Jack Woehr

Other Forth-specific BBS's
- Laboratory Microsystems, Inc.
213-306-3530
StarLink node 9184 on TymNet
PC-Pursuit node calan on TeleNet
SysOp: Ray Duncan
- Knowledge-Based Systems
Supports Fifth
409-696-7055
- Druma Forth Board
512-323-2402
StarLink node 1306 on TymNet
SysOps: S. Suresh, James Martin, Anne Moore
- Harris Semiconductor Board
407-729-4949
StarLink node 9902 on TymNet (toll from Post. St. Lucie)

Non-Forth-specific BBS's with extensive Forth Libraries
- Twit's End (PC Board)
501-771-0114
1200-9600 baud
StarLink node 9858 on TymNet
SysOp: Tommy Apple
- College Corner (PC Board)
206-643-0804
300-2400 baud
SysOp: Jerry Houston
- Psymatic BBS
Sunnyvale, California
408-992-0372
300 – 2400 baud
This is a programmer's board with a

SIGFORTH '91

REGISTRATION

ANNUAL

SIGFORTH CONFERENCE

MARCH 7-9, 1991

COMPLETE & MAIL THIS FORM TO:

SIGFORTH CONFERENCE
THE SOFTWARE CONSTRUCTION CO.
2900B LONGMIRE DRIVE
COLLEGE STATION, TX 77845
(409) 696-5432

PLEASE TYPE OR PRINT CLEARLY

NAME _____

ORGANIZATION _____

ADDRESS _____

CITY _____ STATE _____

ZIP _____ PHONE _____

FAX _____

CONFERENCE FEE:
PAYMENT BY CHECK ONLY PAYABLE TO: SIGFORTH '91

$190	ACM or SIGFORTH MEMBER
$190	GOVERNMENT (AGENCY) _____
$240	NONMEMBER
$50	STUDENT
$50	LATE FEE (AFTER 12/31/90)

HOTEL REGISTRATION:

HYATT REGENCY OF SAN ANTONIO
123 LOSOYA STREET
SAN ANTONIO, TX 78205
PHONE: (512) 222-1234
FAX: (512) 227-4925

large Forth area.

International Forth BBS's

- Melbourne FIG Chapter
  (03) 809-1787 in Australia
  61-3-809-1787 international
  SysOp: Lance Collins
- Forth BBS JEDI
  Paris, France
  33 36 43 15 15
  7 data bits, 1 stop, even parity
- Max BBS *(ForthNet link*)*
  United Kingdom
  0905 754157
  SysOp: Jon Brooks
- Sky Port *(ForthNet link*)*
  United Kingdom
  44-1-294-1006
  SysOp: Andy Brimson
- SweFIG
  Per Alm Sweden
  46-8-71-35751
- NEXUS Servicios de Informacion, S. L.
  Travesera de Dalt, 104-106, Entlo. 4–5
  08024 Barcelona, Spain
  + 34 3 2103355 (voice)
  + 34 3 2147262 (modem)
  SysOps: Jesus Consuegra, Juanma Barranquero
  barran@nexus.nsi.es (preferred)
  barran@nsi.es
  barran (on BIX)

This list was accurate as of October 1990. If you know another on-line Forth resource, please let me know so it can be included in this list. I can be reached in the following ways:

Gary Smith
P. O. Drawer 7680
Little Rock, Arkansas 72217
Telephone: 501-227-7817
GEnie (co-SysOp, Forth RT and Unix RT): GARY-S
Usenet domain.: uunet!wugate!wuarchive!texbell!ark!lrark!gars

**ForthNet is a virtual Forth network that links designated message bases in an attempt to provide greater information distribution to the Forth users served. It is provided courtesy of the SysOps of its various links.*

(Continued from page 34.)

I'd be particularly interested in any problems you devised for your course.

"...a good excuse for looking at pretty pictures!"

I enjoy image processing, too, but what sort of programs could you assign to beginning Forth students along those lines?

To: Mark Smiley
From: Archie Warnock
Subj: Teaching F-PC

"Which of the many ways did you suggest?"

I just let them try things. We'd already done @ and ! when I first suggested this, and they'd go back to it periodically when I'd point out a particularly dangerous construct. Mostly they tried putting zero into various memory locations. I had one pair of guys who really liked trying it with the assembler.

"I'd be particularly interested in any problems you devised for your course."

Mostly, I worked from Jack's notes. I was teaching a non-credit course here at Goddard, and so didn't really make assignments, as such. I basically gave my own notes and then had them work on Jack's problems as a lab.

"I enjoy image processing, too, but what sort of programs could you assign to beginning Forth students along those lines?"

It's an enormously powerful example of vectored execution. Also, defining matrices and images is a good example of using defining words.

To suggest an interesting on-line guest, leave e-mail posted to GARY-S on GEnie (gars on Wetware and the Well), or mail me a note. I encourage anyone with a message to share to contact me via the above or through the offices of the Forth Interest Group.

(Koopman code, continued.)

```
: INIT-TEST  ( ..DS.stuff..  ..RS.stuff..  -- ..DS.stuff.. )
             ( RS:   -- ..RS.stuff.. )
  CR ."  TEST-"
  #STACK @  0<  ABORT" You must specify both DS( and RS(."
  R> R-SAVE !  R> R-SAVE 2+ !          \ Save return address
  #STACK @  REVERSE
  BEGIN  #STACK @  0> WHILE  >R  -1 #STACK +!  REPEAT
  R-SAVE 2+ @ >R  R-SAVE @  >R   ;     \ Restore return address

: FINISH-TEST  ( ..DS.stuff.. -- ..DS.stuff.. ..reversed.RS.stuff.. )
               ( RS:  ..RS.stuff.. -- )
  R> R-SAVE ! R> R-SAVE 2+ !           \ Save return address
  \ Transfer return stack contents onto data stack for later compare
  0 >R
  BEGIN  R> R>  SWAP 1+ >R   DUP $BAD3 = UNTIL
  R> REVERSE
  R-SAVE 2+ @ >R  R-SAVE @  >R         \ Restore return address
  ." -DONE"  -1 #STACK !  ;

\ TEST and DONE use F-TZ specific words to compile a short
\  definition containing the word to be tested, execute that
\  definition, then FORGET it from the dictionary.
\ This borrows a compilation idea from Rick vanNorman's RTX test code
CREATE MARKER  4 ALLOT
: TESTER ;
: TEST:   ( -)
   XHERE  2DUP MARKER 2!  PARAGRAPH +  DUP XDPSEG !  0 XDP !
   XSEG @ -  ['] TESTER >BODY  !
   COMPILE INIT-TEST  ] ;

: ;DONE
   COMPILE FINISH-TEST  COMPILE EXIT
   STATE OFF  TESTER  MARKER 2@  XDP ! XDPSEG !  ;
IMMEDIATE


\ Test ROT for proper operation
  DS( 1111 2222 3333 --
  RS( --
  TEST: ROT ;DONE
  --- )RS
  --- 2222 3333 1111 )DS

\ Test >R for proper operation
  DS( 5555 --
  RS( --
  TEST:  >R  ;DONE
  --- 5555 )RS
  --- )DS

\ Any combination may go between TEST: and ;DONE
  DS( 1111 2222 3333 --
  RS( 7777 2222 9999 --
  TEST:  SWAP R> ROT >R  ;DONE
  --- 7777 2222 3333 )RS
  --- 1111 2222 9999 )DS

\ Null test to be sure it works
  DS( --
  RS( --
  TEST:  ;DONE
  --- )RS
  --- )DS
```

(Continued from page 18.)

Example A

```
: SETC                or     : SETC
  1 2*c DROP  ;               0 -c  ;
```

Example B

```
: CLRC                or     : CLRC
  0 2*c DROP  ;               0 +c  ;
```

Figure Nine. Two different high-level definitions for SETC and CLRC.

row. The words in Figure Three must synthesize the carry to (or the borrow from) the higher order place values with the D+ and/or D- operators. (In T- the phrase DUP R> R> D+ adds -1 to [am ah] if a1 < b1, else it adds zero.)

You can use this symmetry to create arbitrary-precision addition (and subtraction) operators very easily (see Figure Four). To further illustrate the utility of our newly developed capability, some extended-precision, mixed-math operators are created in Figure Five.

Now let's produce some extended-precision shift operators using 2*c and c2/ (Figure Six). Examples of right-shift extended-precision operators are given in Figure Seven. Next, the examples in Figure Eight stretch this concept to produce fast multiple-shift, extended-precision operators.

Conclusion

With the inclusion of these proposed six words in your Forth, your ability to do extended-precision math will be greatly enhanced. A minimal amount of machine-level coding is all that is necessary to give you some powerful Forth words. In fact, only +c, -c, 2*c, and c2/ need to be coded in assembly. SETC and CLRC can be defined in a couple of ways, shown in Figure Nine.

I thought these words to be so useful that I submitted proposals to include them to the January 1990 meeting of the ANSI Forth standard committee. However, they were rejected (although my U2/ did pass). If you think these words are useful and should be included in the standard in some form or fashion, write and submit proposals to ANSI X3J14, Forth Standards Committee, 111 N. Sepulveda Boulevard, Suite 300, Manhattan Beach, California 90266. Time is running short for submitting new proposals to the committee, so speak now while the process is still open.

Douglas Ross designs digital systems for NASA. He has created systems around the NC4000 Forth engine, and a Wideband Transport Frame Formatter using Harris' RTX 2000 with an interactive cmFORTH kernel. As a beta site for the RTX 2010, he will use that chip in both the Command and Data Handling subsystem of the MODIS-T instrument and in a project to build a flight computer core. He wants to develop an ANSI Forth for NASA to facilitate better sharing of Forth applications there.

(Continued from page 10.)

Racine, Wisconsin 53404-3336

Endnotes

1. Intel 80386 microprocessors may also have the division error interrupt return address bug. In a brief test, an 80386-based IBM PS-2 Model 60 was shown to leave the wrong return address, and then to work properly when an 80286 bug fix was used. No technical reference was available, though, to document this as a characteristic of every 80386.

2. *iAPX 86, 88, 186, and 188 User's Manual Programmer's Reference*, page 3-75. The divisor is left unchanged, whether it's in a CPU register or in memory.

3. Intel, page 4-9.

4. Intel, pages 3-26, 3-39, 3-75. Only the 8086 is afflicted with the signed division bug. The 80186 properly handles the full range of valid signed results. The 80286 and 80386 computers I've tried seem to handle the full range of results.

5. *Technical Reference Personal Computer AT*, page 9-9.

6. That's unofficial, though. Intel says the quotient and remainder registers (which initially hold the dividend) are left undefined.

7. Jon Salmon, a student at Northeast Missouri State University, suggested using dummy safe operands to allow a division instruction to continue after return from an error interrupt. He also suggested using a table of opcodes and corresponding instruction lengths to bump a possibly incorrect return address.

8. Four NOPs, which would match the length of the longest possible division instruction, execute in twelve clock cycles. A word-length DIV instruction, for comparison, executes in 144–181 cycles on an 8086, and in 22–26 cycles on an 80286.

(Continued on next page.)

. Robert Berkey. "Positive-Divisor Floored Division," *Forth Dimensions* (XII/1) has many interesting comments about properties of floored and unfloored division, and many practical examples.

Acknowledgments

George Barlow, John Erhart, Bryce Jones, NMSU Computer Lab (Northeast Missouri State University), NMSU Library, Jon Salmon, T&R Electronics.

David Arnold has had some training in engineering school. He is trying to devise methods and equipment that will help him, a handicapped person, to work both at home and in typical workplaces.

(Continued from page 25.)

```
ASSEMBLER \ Invoke vocabulary

CLEAR.TABLES  \ Prepare for DLAB and ULAB tables

ASSEMBLE TEST DLAB ONE 0 # LDA, 80 # LDY, DLAB TWO 300 ,Y
STA, DEY, ULAB TWO BPL, RTS,

ASSEMBLE TRY ULAB ONE JSR, ULAB OUT LDA, PHA, ULAB OUT1 LDA,
ULAB PUT JMP,

ASSEMBLE THIS DLAB PUSH DEX, DEX, DLAB PUT 1 ,X STA, PLA, 0
,X STA, ' NEXT 2+ JMP,

VARIABLE OUT  -2 ALLOT DLAB OUT 1 ALLOT DLAB OUT1 1 ALLOT

ASSEMBLE ADVANCE CLC, ULAB OUT LDA, 1 # ADC, ULAB OUT STA,
ULAB OUT1 LDA, 0 # ADC, ULAB OUT1 STA, ' NEXT 2+ JMP,

: NOTHING ;

: SOMETHING 1+ ;

: ASSEMBLE CALL 0 ,X LDA, ULAB CALL1 FF00 STA, 1 ,X LDA,
ULAB CALL2 FF00 STA, JSR, -2 ALLOT DLAB CALL1 1 ALLOT DLAB
CALL2 1 ALLOT INX, INX, ' NEXT 2+ JMP,

END
```

Figure One. Illustrative application.

ADVERTISERS INDEX

FIG CHAPTERS

The FIG Chapters listed below are currently registered as active with regular meetings. If your chapter listing is missing or incorrect, please contact Anna Brereton at the FIG office's Chapter Desk. This listing will be updated in each issue of *Forth Dimensions*. If you would like to begin a FIG Chapter in your area, write for a "Chapter Kit and Application." Forth Interest Group, **P.O. Box 8231, San Jose, California 95155**

U.S.A.

- **ALABAMA**
  Huntsville Chapter
  Tom Konantz
  (205) 881-6483

- **ALASKA**
  Kodiak Area Chapter
  Ric Shepard
  Box 1344
  Kodiak, Alaska 99615

- **ARIZONA**
  Phoenix Chapter
  4th Thurs., 7:30 p.m.
  Arizona State Univ.
  Memorial Union, 2nd floor
  Dennis L. Wilson
  (602) 381-1146

- **CALIFORNIA**
  Los Angeles Chapter
  4th Sat., 10 a.m.
  Hawthorne Public Library
  12700 S. Grevillea Ave.
  Phillip Wasson
  (213) 649-1428

  North Bay Chapter
  2nd Sat., 10 a.m. Forth, AI
  12 Noon Tutorial, 1 p.m. Forth
  South Berkeley Public Library
  George Shaw (415) 276-5953

  Orange County Chapter
  4th Wed., 7 p.m.
  Fullerton Savings
  Huntington Beach
  Noshir Jesung (714) 842-3032

  Sacramento Chapter
  4th Wed., 7 p.m.
  1708-59th St., Room A
  Bob Nash
  (916) 487-2044

  San Diego Chapter
  Thursdays, 12 Noon
  Guy Kelly (619) 454-1307

  Silicon Valley Chapter
  4th Sat., 10 a.m.
  H-P Cupertino
  John Hall (415) 532-1115

  Stockton Chapter
  Doug Dillon (209) 931-2448

- **COLORADO**
  Denver Chapter
  1st Mon., 7 p.m.
  Clifford King (303) 693-3413

- **FLORIDA**
  Orlando Chapter
  Every other Wed., 8 p.m.
  Herman B. Gibson
  (305) 855-4790

  Tampa Bay Chapter
  1st Wed., 7:30 p.m.
  Terry McNay (813) 725-1245

- **GEORGIA**
  Atlanta Chapter
  3rd Tues., 7 p.m.
  Emprise Corp., Marietta
  Don Schrader (404) 428-0811

- **ILLINOIS**
  Cache Forth Chapter
  Oak Park
  Clyde W. Phillips, Jr.
  (708) 713-5365

  Central Illinois Chapter
  Champaign
  Robert Illyes (217) 359-6039

- **INDIANA**
  Fort Wayne Chapter
  2nd Tues., 7 p.m.
  I/P Univ. Campus
  B71 Neff Hall
  Blair MacDermid
  (219) 749-2042

- **IOWA**
  Central Iowa FIG Chapter
  1st Tues., 7:30 p.m.
  Iowa State Univ.
  214 Comp. Sci.
  Rodrick Eldridge
  (515) 294-5659

  Fairfield FIG Chapter
  4th Day, 8:15 p.m.
  Gurdy Leete (515) 472-7077

- **MARYLAND**
  MDFIG
  3rd Wed., 6:30 p.m.
  JHU/APL, Bldg. 1
  Parsons Auditorium
  Mike Nemeth (301) 262-8140
  (eves.)

- **MASSACHUSETTS**
  Boston Chapter
  3rd Wed., 7 p.m.
  Honeywell
  300 Concord, Billerica
  Gary Chanson (617) 527-7206

- **MICHIGAN**
  Detroit/Ann Arbor Area
  Bill Walters
  (313) 731-9660
  (313) 861-6465 (eves.)

- **MINNESOTA**
  MNFIG Chapter
  Minneapolis
  Fred Olson
  (612) 588-9532

- **MISSOURI**
  Kansas City Chapter
  4th Tues., 7 p.m.
  Midwest Research Institute
  MAG Conference Center
  Linus Orth (913) 236-9189

  St. Louis Chapter
  1st Tues., 7 p.m.
  Thornhill Branch Library
  Robert Washam
  91 Weis Drive
  Ellisville, MO 63011

- **NEW JERSEY**
  New Jersey Chapter
  Rutgers Univ., Piscataway
  Nicholas Lordi
  (201) 338-9363

- **NEW MEXICO**
  Albuquerque Chapter
  1st Thurs., 7:30 p.m.
  Physics & Astronomy Bldg.
  Univ. of New Mexico
  Jon Bryan (505) 298-3292

- **NEW YORK**
  Long Island Chapter
  3rd Thurs., 7:30 p.m.
  Brookhaven National Laboratory
  AGS dept., bldg. 911, lab rm. A-202
  Irving Montanez
  (516) 282-2540

Rochester Chapter
Monroe Comm. College
Bldg. 7, Rm. 102
Frank Lanzafame
716) 482-3398

- **OHIO**
  Cleveland Chapter
  4th Tues., 7 p.m.
  Chagrin Falls Library
  Gary Bergstrom
  216) 247-2492

- **Columbus FIG Chapter**
  4th Tues.
  Kal-Kan Foods, Inc.
  5115 Fisher Road
  Terry Webb
  614) 878-7241

  Dayton Chapter
  2nd Tues. & 4th Wed., 6:30 p.m.
  CFC. 11 W. Monument Ave. #612
  Gary Ganger (513) 849-1483

- **OREGON**
  Willamette Valley Chapter
  4th Tues., 7 p.m.
  Linn-Benton Comm. College
  Pann McCuaig (503) 752-5113

- **PENNSYLVANIA**
  Villanova Univ. Chapter
  1st Mon., 7:30 p.m.
  Villanova University
  Dennis Clark
  (215) 860-0700

- **TENNESSEE**
  East Tennessee Chapter
  Oak Ridge
  3rd Wed., 7 p.m.
  Sci. Appl. Int'l. Corp., 8th Fl.
  800 Oak Ridge Turnpike
  Richard Secrist
  (615) 483-7242

- **TEXAS**
  Austin Chapter
  Matt Lawrence
  PO Box 180409
  Austin, TX 78718

  Dallas Chapter
  4th Thurs., 7:30 p.m.
  Texas Instruments
  13500 N. Central Expwy.
  Semiconductor Cafeteria
  Conference Room A
  Clif Penn (214) 995-2361

Houston Chapter
3rd Mon., 7:30 p.m.
Houston Area League of PC Users
1200 Post Oak Rd.
(Galleria area)
Russell Harris
(713) 461-1618

- **VERMONT**
  Vermont Chapter
  Vergennes
  3rd Mon., 7:30 p.m.
  Vergennes Union High School
  RM 210, Monkton Rd.
  Hal Clark (802) 453-4442

- **VIRGINIA**
  First Forth of Hampton Roads
  William Edmonds
  (804) 898-4099

  Potomac FIG
  D.C. & Northern Virginia
  1st Tues.
  Lee Recreation Center
  5722 Lee Hwy., Arlington
  Joseph Brown
  (703) 471-4409
  E. Coast Forth Board
  (703) 442-8695

  Richmond Forth Group
  2nd Wed., 7 p.m.
  154 Business School
  Univ. of Richmond
  Donald A. Full
  (804) 739-3623

- **WISCONSIN**
  Lake Superior Chapter
  2nd Fri., 7:30 p.m.
  1219 N. 21st St., Superior
  Allen Anway (715) 394-4061

INTERNATIONAL

- **AUSTRALIA**
  Melbourne Chapter
  1st Fri., 8 p.m.
  Lance Collins
  65 Martin Road
  Glen Iris, Victoria 3146
  03/889-2600
  BBS: 61 3 809 1787

  Sydney Chapter
  2nd Fri., 7 p.m.
  John Goodsell Bldg., RM LG19
  Univ. of New South Wales
  Peter Tregeagle
  10 Binda Rd.
  Yowie Bay 2228
  02/524-7490
  Usenet
  tedr@usage.csd.unsw.oz

- **BELGIUM**
  Belgium Chapter
  4th Wed., 8 p.m.
  Luk Van Loock
  Lariksdreff 20
  2120 Schoten
  03/658-6343

  Southern Belgium Chapter
  Jean-Marc Bertinchamps
  Rue N. Monnom, 2
  B-6290 Nalinnes
  071/213858

- **CANADA**
  BC FIG
  1st Thurs., 7:30 p.m.
  BCIT, 3700 Willingdon Ave.
  BBY, Rm. 1A-324
  Jack W. Brown
  (604) 596-9764
  BBS (604) 434-5886

  Northern Alberta Chapter
  4th Sat., 10a.m.-noon
  N. Alta. Inst. of Tech.
  Tony Van Muyden
  (403) 486-6666 (days)
  (403) 962-2203 (eves.)

  Southern Ontario Chapter
  Quarterly, 1st Sat., Mar., Jun., Sep., Dec., 2 p.m.
  Genl. Sci. Bldg., RM 212
  McMaster University
  Dr. N. Solntseff
  (416) 525-9140 x3443

- **ENGLAND**
  Forth Interest Group-UK
  London
  1st Thurs., 7 p.m.
  Polytechnic of South Bank
  RM 408
  Borough Rd.
  D.J. Neale
  58 Woodland Way
  Morden, Surry SM4 4DS

- **FINLAND**
  FinFIG
  Janne Kotiranta
  Arkkitehdinkatu 38 c 39
  33720 Tampere
  +358-31-184246

- **HOLLAND**
  Holland Chapter
  Vic Van de Zande
  Finmark 7
  3831 JE Leusden

- **ITALY**
  FIG Italia
  Marco Tausel
  Via Gerolamo Forni 48
  20161 Milano
  02/435249

- **JAPAN**
  Tokyo Chapter
  3rd Sat. afternoon
  Hamacho-Kaikan, Chuoku
  Toshio Inoue
  (81) 3-812-2111 ext. 7073

- **REPUBLIC OF CHINA**
  R.O.C. Chapter
  Chin-Fu Liu
  5F, #10, Alley 5, Lane 107
  Fu-Hsin S. Rd. Sec. 1
  TaiPei, Taiwan 10639

- **SWEDEN**
  SweFIG
  Per Alm
  46/8-929631

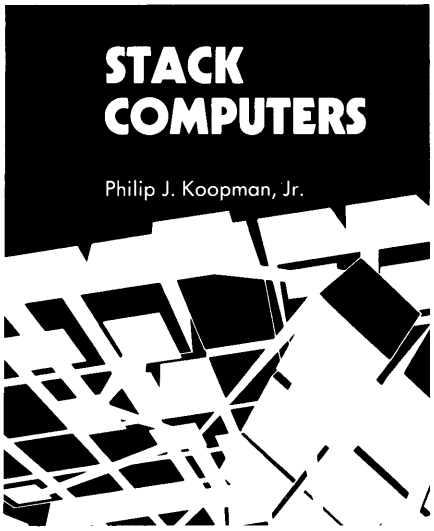
- **SWITZERLAND**
  Swiss Chapter
  Max Hugelshofer
  Industrieberatung
  Ziberstrasse 6
  8152 Opfikon
  01 810 9289

- **WEST GERMANY**
  German FIG Chapter
  Heinz Schnitter
  Forth-Gesellschaft C.V.
  Postfach 1110
  D-8044 Unterschleissheim
  (49) (89) 317 3784
  Munich Forth Box:
  (49) (89) 725 9625 (telcom)

SPECIAL GROUPS

- **NC4000 Users Group**
  John Carpenter
  1698 Villa St.
  Mountain View, CA 94041
  (415) 960-1256 (eves.)

NEW FROM THE FORTH INTEREST GROUP

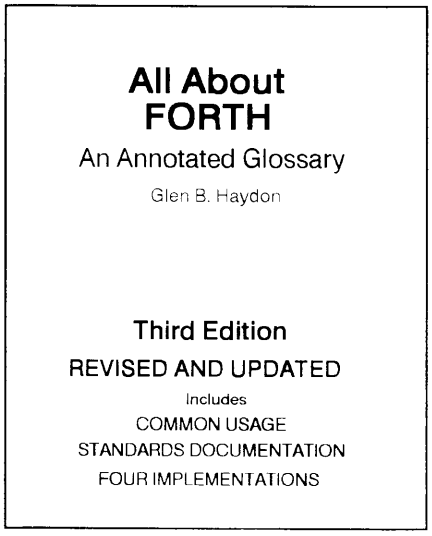


STACK COMPUTERS
the new wave

by Philip J. Koopman, Jr.

This book presents an alternative to Complex Instruction Set Computers (CISC) and Reduced Instruction Set Computers (RISC) by showing the strengths and weaknesses of stack machines.

$62.00



ALL ABOUT FORTH
the 3rd Edition

by Glen B. Haydon

An Annotated glossary of most Forth words in common usage, including Forth-79, Forth-83, F83, F-PC, MVP-FORTH. Implementation examples in high-level Forth and/or 8086/8088 assembler, and useful commentary, are given for each entry.

$90.00

NOW AVAILABLE!
SEE ORDER FORM INSIDE

Forth Interest Group
P.O.Box 8231
San Jose, CA 95155

Second Class
Postage Paid at
San Jose, CA